Optimisation de tournées de véhicules dans le cadre de la logistique inverse : modélisation et résolution par des méthodes hybrides

Emilie Grellier

UNIVERSITÉ DE NANTES
FACULTÉ DES SCIENCES ET DES TECHNIQUES

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATÉRIAUX

Année 2008

Optimisation des tournées de véhicules dans le cadre de la logistique inverse : modélisation et résolution par des méthodes hybrides

---

Thèse de doctorat de l'université de Nantes

Discipline : Informatique Appliquée

Spécialité : Recherche Opérationnelle

Présentée
et soutenue publiquement par

Émilie Grellier

Le 30 janvier 2008 à l'École Nationale Supérieure
des Techniques Industrielles et des Mines de Nantes

Devant le jury ci-dessous :

Président	:	Christian Prins, Professeur, Université de Technologie de Troyes
Rapporteurs	:	Christine Solnon, Maître de Conférences, HdR, Université Lyon 1
		Frédéric Semet, Professeur, Université de Valenciennes et du Hainaut-Cambrésis
Examinateur	:	Xavier Gandibleux, Professeur, Université de Nantes
Directeur de thèse	:	Narendra Jussien, Professeur, École des Mines de Nantes
Co-encadrant	:	Pierre Dejax, Professeur, École des Mines de Nantes

Équipes d'accueil	:	SLP – IRCCyN
		Contraintes – LINA
Laboratoires d'accueil	:	Institut de Recherche en Communications et Cybernétique de Nantes
		Laboratoire d'Informatique de Nantes Atlantique
Composante de rattachement du directeur de thèse	:	École des Mines de Nantes

N° ED 366-351

Table des matières

Table des figures

Liste des tableaux

Introduction

Le contexte

Nous sommes actuellement dans une conjoncture où le transport routier se trouve de plus en plus mis à mal. En effet les récents débats sur la politique à mener pour améliorer notre impact sur l'écologie ainsi que les nombreuses augmentations du prix du pétrole laissent à penser qu'il est nécessaire de minimiser au maximum les différents déplacements de nos véhicules.

Les recherches sur l'optimisation de la logistique, et plus particulièrement les recherches sur la construction des tournées de véhicules, vont dans le sens de nos besoins industriels et humains. Les constructions de tournées sont utilisées tant pour le transport de marchandises, de déchets, de courriers que pour le transport de personnes. Le réseau logistique dans lequel se fait cette optimisation se divise en plusieurs niveaux. Plusieurs demandes de transport sont envisageables au sein et entre ces niveaux. Certains réseaux se distinguent par la particularité de leurs caractéristiques telles que la présence d'une flotte de véhicules pour satisfaire les demandes dont la capacité est homogène ou hétérogène, la présence de dépôts intermédiaires afin de recharger ou décharger les véhicules en cours de tournées, la présence de contraintes d'accessibilité sur les différents sites à visiter, la nécessité de couvrir le réseau via les arcs ou les sommets. D'autres se distinguent par les caractéristiques des demandes telles que une demande de collecte ou livraison, une demande pouvant être livrée par plusieurs véhicules (cas des demandes pouvant être divisées), la présence de fenêtres de visite au sein desquelles les demandes doivent être effectuées. Les constructions de tournées peuvent avoir plusieurs objectifs. Nous pouvons notamment citer la minimisation des coûts ou encore la minimisation du nombre de véhicules utilisés. L'optimisation des tournées permet de réaliser des accroissements de productivité (réduction des coûts) comme l'amélioration de l'organisation et de la qualité de service à la clientèle.

Le secteur de la logistique inverse est devenu un domaine très porteur. La logistique inverse considère les flux dits inverses (allant des consommateurs vers le producteur), en opposition aux flux dits traditionnels (allant du producteur vers les consommateurs). Les flux inverses peuvent être des flux de retours de produits en cas de non satisfaction, dans le cadre d'un service après-vente ou encore lorsque celui-ci arrive en fin de vie. Les retours peuvent également être des flux de matériaux de conditionnement dans le but de les réutiliser ou bien de les recycler. La mise en place d'une telle organisation peut être motivée par plusieurs enjeux : économique, écologique, législatif ou d'amélioration du service client.

Malgré l'utilisation de véhicules au sein des réseaux de logistique inverse, elle contribue au recyclage de nombreux produits très polluants, à la récupération des matières premières et à la prise en compte des produits retournés par les consommateurs ce qui fait son succès. Ce genre de réseau très prometteur couplé à une optimisation des déplacements effectués dans celui-ci est une bonne manière d'aller dans le sens du respect de l'environnement et du développement durable, tout en minimisant les frais liés à l'utilisation des véhicules. C'est pour toutes ces raisons que nous avons étudié un problème de construction de tournées dans un contexte de logistique inverse.

Les objectifs

Dans cette thèse, les objectifs sont doubles. En effet nous pouvons diviser nos objectifs en deux parties qui sont : les objectifs *problème* et les objectifs *techniques*.

Les objectifs *problème* résident essentiellement dans la mise en place d'un réseau assez générique pour pouvoir être utilisé dans divers contextes : logistique traditionnelle de livraison, logistique traditionnelle de collecte,

distribution et collecte de courriers, *etc.* Nous étudions un réseau constitué d'un dépôt et de n magasins. Plusieurs véhicules constituent la flotte disponible au dépôt pour effectuer les livraisons de produits et pour collecter les retours des magasins. Un éventail large d'options telles que la variation des différents coûts, la possibilité ou non de livrer un même client en plusieurs fois, la gestion ou non des stocks des magasins[1] par le dépôt central est proposée. Nous avons voulu travailler sur un problème issu de la logistique inverse mais le réseau étudié s'adapte à d'autres problèmes dans un tout autre contexte. De plus le problème traité est original. Il est modulable et peut ainsi être traité avec diverses contraintes afin de permettre une réutilisation. Ainsi l'objectif principal est de trouver selon les différentes options du réseau certaines préconisations quant à la méthode à appliquer sur le problème.

Les objectifs *techniques* résident quant à eux dans l'emploi de techniques de programmation par contraintes au sein des méthodes de résolution via les méthodes hybrides. La programmation par contraintes permet de résoudre des problèmes combinatoires tout comme la recherche opérationnelle. La complémentarité des deux techniques via les méthodes hybrides a prouvé son efficacité sur plusieurs problèmes combinatoires. Ainsi les méthodes hybrides semblent une voie prometteuse pour résoudre le problème de construction de tournées de véhicules. Nous souhaitons donc dans un premier temps valider la cohérence de l'emploi de telles méthodes sur des problèmes comme celui que nous traitons. Puis dans un second temps il s'agira de comparer les performances des méthodes n'employant que des techniques issues de la recherche opérationnelle avec celles des méthodes hybrides. De plus diverses techniques de résolution seront testées : heuristique, métaheuristique et exacte.

L'organisation de la thèse

Cette thèse se découpe en trois parties séparant ici l'aspect définition du problème des méthodes de résolution utilisées et des résultats obtenus.

La première partie est consacrée à la description du problème de tournées de véhicules que nous traitons et à son positionnement dans la littérature. Ce problème est décrit dans son contexte : la logistique inverse que nous définirons dans le chapitre 1. Pour cela, nous verrons dans un premier temps plusieurs définitions de la logistique inverse, les différentes finalités des produits récupérés dans la logistique inverse et les différents intérêts à la mise en place d'un tel réseau logistique. Puis, nous introduirons les problèmes de construction de tournées qui constituent le fondement du problème étudié. Et enfin, nous caractériserons le problème traité et proposerons deux modèles mathématiques pour les deux versions du problème (en juste à temps et avec gestion des stocks[1]). Le chapitre 2 permet de faire un état de la littérature sur les trois domaines principaux de cette thèse. Nous commencerons par faire une revue de la littérature de la logistique inverse sur les trois niveaux de la planification hiérarchisée : stratégique, tactique et opérationnel. Puis nous verrons les travaux qui concernent les problèmes de construction de tournées avec plusieurs variantes : avec fenêtres de visite, avec gestion des stocks, avec collecte et livraison et nous verrons les travaux ayant trait au problème étudié. Nous terminons l'état de l'art par un échantillon de travaux réalisés sur des problèmes de transport et qui ont été résolus par le biais de méthodes hybrides. Le chapitre 3 permet de définir l'ensemble des instances que nous avons utilisées pour ce problème. Tout d'abord, nous définissons les instances que nous avons générées aléatoirement avec comme base de travail celles proposées par Solomon (1987) [139] pour le *VRPTW*[2]. Puis, nous présentons les instances proposées par la société alfaplan Management Software & Consulting GmbH qui seront traitées grâce à des méthodes heuristiques.

La seconde partie de cette thèse est consacrée à la résolution du problème traité. Elle constitue une grande partie de notre contribution. Nous commencerons dans le chapitre 4 par les méthodes heuristiques (construction et améliorations). Nous verrons les différentes techniques de construction de solution étudiées. Puis nous verrons l'ensemble des méthodes d'améliorations et les différentes combinaisons de celles-ci. Nous finirons par étudier les différents résultats obtenus et nous conclurons sur la meilleure méthode heuristique trouvée pour le problème en juste à temps et celui avec gestion des stocks. Le chapitre 5 explique quant à lui comment nous avons résolu notre problème grâce à une métaheuristique appelée GRASP[3] [61]. Nous commencerons par expliquer le principe de cette méthode et les différents travaux sur ce sujet dans la littérature. Nous présenterons ensuite comment

[1] Lorsque le dépôt central ne gère pas les stocks des magasins le problème sera alors dans sa version en *"juste à temps"* et lorsqu'au contraire les stocks des magasins sont gérés par le dépôt central le problème sera alors dans sa version appelée *"avec gestion des stocks"*.

[2] Vehicle Routing Problem with Time Windows

[3] Greedy Randomized Adaptive Search Procedure

nous appliquons le GRASP au problème traité tout d'abord en utilisant une recherche locale dite *"classique"* (utilisant des techniques de recherche opérationnelle) au sein de la métaheuristique puis en utilisant une recherche locale dite "hybride" (utilisant des techniques de programmation par contraintes). Le chapitre 6 montre comment nous avons modélisé le problème étudié par un modèle de partitionnement afin de pouvoir utiliser la technique de génération de colonnes [71], [72] suivie d'un *Branch and Bound* [102]. Nous commencerons ce chapitre en rappelant le principe de la technique de génération de colonnes puis nous exposerons les modèles de partitionnement correspondant aux deux versions du problème et nous verrons comment résoudre le sous-problème de plus court chemin sous contraintes de ressources avec différentes techniques : recherche taboue [73] et [74], programmation dynamique et programmation par contraintes.

La troisième partie sera consacrée à des études comparatives des différents résultats obtenus. Le chapitre 7 comparera les résultats selon les différentes catégories d'instances exposées dans le chapitre 3 et selon les méthodes de résolution utilisées. Puis le chapitre 8 offrira des préconisations selon les applications étudiées et exposera les résultats obtenus sur les instances réelles.

Nous terminerons cette thèse par différentes conclusions et perspectives de travaux futurs.

Par ces travaux nous modélisons et résolvons un problème original et générique. Ce problème, permettant l'ajout ou le retrait de plusieurs options, a été résolu par le biais de méthodes heuristiques, métaheuristiques et par génération de colonnes suivi de *Branch and Bound*. Les diverses méthodes d'amélioration et combinaisons de méthodes d'amélioration ont été éprouvées afin d'en identifier la ou les meilleures. La métaheuristique *GRASP* ainsi que la technique de génération de colonnes ont pu être comparées sous diverses approches de résolution faisant appel ou non à la programmation par contraintes. L'ensemble des résultats apporte des préconisations quant à la méthode à employer selon la physionomie du réseau étudié.

Pour résoudre le problème par les méthodes hybrides nous avons utilisé le solveur de contraintes `CHOCO`[4]. Pour la résolution exacte du problème nous nous sommes servis de `XPRESS`. L'ensemble des méthodes de résolution a été développé en `JAVA`.

Le problème dans son contexte a été présenté au cours de la conférence ILS'06 [78]. Les travaux effectués sur les méthodes de construction et améliorations ont fait l'objet d'un rapport technique à l'école des Mines de Nantes [75]. La résolution du problème par métaheuristique a été présentée à la conférence nationale ROADEF'07 [77] et à la conférence internationale MIC'07 [76]. Enfin une approche de résolution par génération de colonnes a été proposée à la conférence internationale Odysseus'06 [79].

[4] http ://choco-solver.net

Première partie

Description du problème de construction de tournées de véhicules dans un contexte de logistique inverse et son positionnement

Chapitre 1

Description du problème de construction de tournées dans la logistique inverse

Les problèmes de tournées de véhicules sont très étudiés dans la littérature. De nombreux travaux traitent de ces problèmes dans leur forme classique (une étude approfondie de ces problèmes a été faite par Toth et Vigo en 2002 [153]), mais dans beaucoup de cas ces problèmes sont particulièrement contraints. Nous pouvons citer de façon non exhaustive les différents cas suivants : présence de fenêtres de visite (Solomon en 1995 [140], Lee *et al.*, 2003 [105]), présence de dépôts intermédiaires (Bard *et al.*, 1998 [6]), présence de différents types d'actions (collectes *et* livraisons : Savelsbergh et Sol, 1995 [136]), *etc.*

Nous étudions le problème de construction de tournées dans le cadre de la logistique inverse. Dans ce contexte d'étude, le problème de construction de tournées de véhicules que nous traitons possède des caractéristiques particulières. Ainsi dans ce chapitre, nous allons tout d'abord décrire ce qu'est la logistique inverse, puis les différents problèmes de construction de tournées et enfin nous verrons le réseau dans lequel nous allons travailler dans la suite du document.

1.1 La logistique inverse

La logistique inverse, comme son nom l'indique, se réfère aux activités de logistique d'une organisation mais dans un sens inversé à ce qu'il peut être dans la logistique traditionnelle. La logistique que nous nommons traditionnelle est celle qui va du système productif vers le consommateur. La logistique inverse, est un concept émergent qui, depuis une dizaine d'années, se rencontre dans la littérature sous différents termes : logistique inversée, *reverse logistics*, gestion de la récupération des produits, logistique à rebours, logistique négative, *etc.*

La logistique inverse est un thème très étudié en marketing[1]. Une autre vision intéressante est celle des industriels eux-mêmes, ainsi un article de "Stratégie Logistique" [33] laisse les dirigeants de plusieurs entreprises définir ce qu'est pour eux la logistique inverse et les apports que cela contribue à développer au sein de leur entreprise.

Plusieurs définitions de ce concept ont pu être trouvées. Nous allons, dans la suite de cette partie en donner quelques-unes, et tenter d'unifier celles-ci afin d'en donner notre conception.

Voyons tout d'abord deux définitions assez générales du concept. Thierry *et al.* (1995) [150] l'appellent la gestion de la récupération des produits (*"product recovery management"*), et la définissent comme :

> *"la gestion des produits, des composants et des matériels usés ou éliminés qui tombent sous la responsabilité de la compagnie manufacturière. L'objectif de la gestion de la récupération des produits est de retirer le maximum de valeur économique raisonnablement possible, tout en réduisant la quantité ultime des déchets."*

Une deuxième proposition à citer est celle donnée par Beaulieu *et al.* (1999) [10] qui définit le concept de logistique inverse sous le terme de logistique à rebours par :

> *"un ensemble d'activités de gestion visant la réintroduction d'actifs secondaires dans des filières à valeur ajoutée."*

[1] Nous invitons le lecteur qui veut en savoir plus à se référer à l'article de Canel-Depitre (2004) [25], qui dans ses travaux s'attache à déceler l'intérêt du consommateur et du citoyen à voir se développer une logistique productive reposant sur la logistique inverse.

En regardant ces deux définitions, on constate qu'elles s'accordent sur un des buts de la logistique inverse qui est de retrouver de la valeur aux produits récupérés.

D'autres auteurs ont cherché à définir la logistique inverse en partant de la définition de la logistique traditionnelle. Citons ainsi Rogers et Tibben-Lembke (1998) [130] qui utilisent la définition fournie par le *"Council of Logistics Management"* :

> *"Processus de planification, de mise en œuvre, de contrôle du flux efficace et rentable de matières premières, de stocks de produits finis et des informations associées à partir du produit de consommation jusqu'au point d'origine dans le but de récupérer la valeur ou d'éliminer de façon appropriée les produits usagés".*

Enfin, une dernière définition qui nous semble intéressante de noter est celle donnée par Chouinard (2003) [29], qui qualifie la logistique inverse ainsi :

> *"Elle consiste à récupérer des biens du circuit commercial ou du consommateur même, de les orienter vers une nouvelle étape de leur existence et de les traiter dans le but d'en retirer le maximum de valeur en cherchant à les réintégrer sur le marché ou de les éliminer proprement. Par son champ d'action, on voudra assurer, entre autres, la gestion et la planification des activités de collecte, d'évaluation, de tri, de désassemblage, de redistribution de même que la gestion des stocks de produits neufs, récupérés et valorisés dans le but de réorienter les produits récupérés de manière efficiente dans leur cycle de vie".*

Parmi l'ensemble de ces définitions, nous considérons que la dernière proposée par Chouinard (2003) [29], résume bien la conception que nous avons de la logistique inverse.

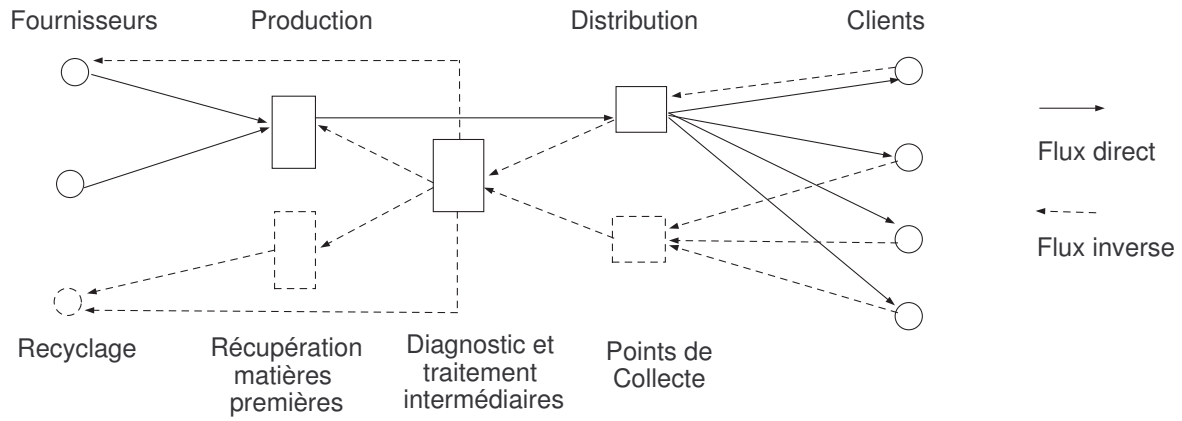
La logistique inverse comprend trois grandes familles de produits : les produits défectueux, le service après vente et les produits pouvant être réutilisés à des fins environnementales (emballages, déchets, produits en fin de vie, DLC[2] dépassée, *etc.*).

Par la logistique inverse souvent on cherche à mieux satisfaire le consommateur. Mieux le satisfaire, c'est lui permettre de rapporter le produit en début de cycle s'il ne lui convient pas et de l'éliminer en fin de vie. Il n'est pas forcément nécessaire dans un réseau de logistique inverse que les retours arrivent au point de départ du réseau de logistique traditionnelle. Ils peuvent effectivement arriver chez un tiers : déchetterie, centre de retraitement, *etc.*

Nous pouvons ainsi utiliser le schéma 1.1 tiré de celui proposé par Bostel *et al.* (2005) [20] pour représenter un réseau logistique global (traditionnel et inverse).

Certains secteurs ont déjà, par nécessité, intégré la logistique inverse dans leur réseau logistique. Nous

Figure 1.1 *Schéma d'un réseau logistique global : incluant les flux directs et indirects*



pouvons ainsi citer : la vente par correspondance, les services de location, *etc.* Avec 35% de retour en moyenne les sociétés d'achats sur catalogue ont dû nécessairement se préoccuper de la mise en place d'un tel système. De plus, le développement des achats par Internet risque de contribuer à l'expansion de la logistique inverse. Il en

[2]Date Limite de Consommation.

est de même pour les sociétés vendant des produits à forte valeur ajoutée (automobile par exemple). Ainsi, on constate que la logistique inverse est un concept novateur et en expansion.

1.1.1 Les différents traitements des produits récupérés et les types de retour

Suite à la récupération des produits dans le cadre de la logistique inverse, plusieurs traitements sont possibles. Thierry *et al.* (1995) [150] classent ces différents traitements en cinq catégories :

- réparation ;
- reconditionnement ;
- réassemblage ;
- récupération de composants (cannibalisation) ;
- recyclage des produits utilisés ou de leurs composants.

Fleischmann (2001) [66] distingue, quant à lui, cinq catégories de retours de produits :

- retours de produits inutilisés ;
- retours commerciaux ;
- retours de produits sous garantie ;
- rebuts et produits dérivés des activités du réseau ;
- emballage.

1.1.2 Les intérêts de la logistique inverse

Les intérêts de structurer une chaîne logistique inverse au sein d'une entreprise sont multiples. Toutefois nous pouvons en considérer quatre.

Le premier pouvant être cité est la conscience environnementale du consommateur. En effet, la prise de conscience envers l'environnement de la part de la société en général favorise et justifie les activités en logistique inverse. Le fait d'être vu par le consommateur comme une entreprise *verte* peut par ailleurs être très vendeur.

Une seconde motivation à l'élaboration d'un tel système est la prise en compte, ou le devancement, des réglementations sur le recyclage et la protection de l'environnement. Nous pouvons citer ici le récent décret sur la gestion des DEEE[3]. Grâce au décret n°2005-829 du 20 juillet 2005, lors de la vente d'un équipement électrique ou électronique ménager, le distributeur reprend gratuitement, ou fait reprendre pour son compte, les équipements électriques et électroniques usagés que lui cède le consommateur, dans la limite de la quantité et du type d'équipement vendu. Ensuite tous les producteurs d'équipements électriques et électroniques doivent pourvoir à la collecte de ces déchets chez leurs différents distributeurs.

Un troisième aspect motivant l'utilisation de la logistique inverse est l'intérêt économique de la réutilisation des produits. En effet, en réutilisant certaines parties de produits, l'entreprise et le client font des économies. Krikke *et al.* (1999) [101] montrent dans leur étude sur le recyclage de moniteurs d'ordinateurs qu'un gain de 25% peut être effectué grâce à la réutilisation de certaines parties.

Enfin le quatrième aspect concerne l'amélioration du niveau et de la qualité de service qui permet le retour des produits par les clients.

1.2 Les problèmes de construction de tournées de véhicules

Nous avons décidé de focaliser nos travaux concernant la logistique inverse sur les problèmes de construction de tournées de transport dont l'optimisation peut être un facteur déterminant dans la prise en compte des doubles flux. Dans cette partie nous allons rappeler les bases des différents problèmes de tournées de véhicules.

1.2.1 Le problème de tournées de véhicules

Le problème de construction de tournées de véhicules connu sous le nom de *Vehicle Routing Problem (VRP)* est une extension du problème du voyageur de commerce, connu également sous le nom de *Travelling Salesman Problem (TSP)*. Nous allons donc dans un premier temps définir le problème du voyageur de commerce, puis

[3] Déchets d'Équipements Électriques et Électroniques.

nous verrons son extension : le problème de construction de tournées.

Le problème du voyageur de commerce

Soit $G = (V, A)$ un graphe où V représente l'ensemble de n sommets et A l'ensemble des arcs (si le graphe est orienté) ou arêtes. Chaque arête ou arc (i, j) du graphe possède un coût noté : c_{ij}. L'objectif de ce problème est de trouver un cycle ou circuit de coût minimum, visitant l'ensemble des n sommets du graphe. Autrement dit, il s'agit de trouver un cycle ou circuit Hamiltonien de coût minimum. Ce problème est *NP-difficile* [92].

Une formulation linéaire de ce problème a été proposée par Dantzig *et al.* (1954) [40]. Soient x_{ij} une variable binaire égale à 1 si l'arc (i, j) est utilisé dans le trajet et à 0 sinon, c_{ij} le coût de parcours de l'arc (i, j). On peut alors formaliser le problème de TSP comme suit :

$$\text{Min} : z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1.1}$$

Sous les contraintes :

$$\forall\, j \in V \qquad \sum_{i \in V} x_{ij} = 1 \tag{1.2}$$

$$\forall\, i \in V \qquad \sum_{j \in V} x_{ij} = 1 \tag{1.3}$$

$$\forall\, S \subset V\ ;\ 2 \leq |S| \leq n-2 \qquad \sum_{i,j \in S} x_{ij} \leq |S| - 1 \tag{1.4}$$

$$\forall\, i \in V\ \forall\, j \in V \qquad x_{ij} \in \{0, 1\} \tag{1.5}$$

Les contraintes 1.2 et 1.3 permettent d'assurer que le voyageur n'entre et ne sorte qu'une seule fois par sommet. La contrainte 1.4 élimine la formation de sous-tour au sein d'un voyage. L'objectif de ce problème est donc de minimiser les coûts de transport liés aux arcs visités par le circuit.

Le problème d'élaboration de tournées de véhicules

Le problème de construction de tournées de véhicules (*"Vehicle Routing Problem" (VRP)* en anglais) est une généralisation du problème de *TSP* avec plusieurs voyageurs qui seront appelés véhicules. Le but est de visiter tous les sommets d'un graphe à l'aide d'une flotte de véhicules qui partent et arrivent tous au dépôt. Nous pouvons ainsi définir le graphe $G = (V, A)$, où $V = \{0, \ldots, n\}$ correspond à l'ensemble des $n + 1$ sommets du graphe où 0 représente le dépôt. Chaque client i appartenant à $V \setminus \{0\}$, a une demande de produit d_i qui correspond à la quantité de produit qu'il faut lui livrer ou collecter. Une flotte de M véhicules tous identiques de capacité Q est disponible. L'objectif du VRP est de trouver M tournées (partant et revenant au dépôt) afin que tous les sommets soient visités une unique fois tout en minimisant le coût total de transport et en respectant la capacité de stockage des véhicules.

Ce problème étant une extension du TSP, il est donc *NP-difficile*.

Pour la modélisation, nous allons utiliser la formulation fournie par Fisher et Jaikumar (1978, 1981) [64], [65]. Pour cette formulation, définissons la variable binaire x_{ij}^k qui indique si j est immédiatement visité après i dans la tournée k. Nous avons également la variable binaire y_i^k qui est égale à 1 si le véhicule k effectue le

service chez le client i. Ainsi nous pouvons écrire :

$$\text{Min} : z = \sum_{k=1}^{M} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}^{k} \tag{1.6}$$

Sous les contraintes :

$$\forall\, k \in (1, \ldots, n) \qquad \sum_{i=1}^{n} d_i y_i^k \leq Q \tag{1.7}$$

$$\forall\, i \in (1, \ldots, n) \qquad \sum_{k=1}^{M} y_i^k = 1 \tag{1.8}$$

$$\sum_{k=1}^{M} y_1^k = M \tag{1.9}$$

$$\forall\, j \in (1, \ldots, n) \quad \forall\, k \in (1, \ldots, M) \qquad \sum_{i=1}^{n} x_{ij}^k = y_j^k \tag{1.10}$$

$$\forall\, i \in (1, \ldots, n) \quad \forall\, k \in (1, \ldots, M) \qquad \sum_{j=1}^{n} x_{ij}^k = y_i^k \tag{1.11}$$

$$\forall\, S \subset V\ ;\ 2 \leq |S| \leq n-2 \quad \forall\, k \in (1, \ldots, M) \qquad \sum_{i,j \in S} x_{ij}^k \leq |S| - 1 \tag{1.12}$$

$$\forall\, i, j \in V\ i \neq j \quad \forall\, k \in (1, \ldots, M) \qquad x_{ij}^k \in \{0, 1\} \tag{1.13}$$

$$\forall\, i \in V \quad \forall\, k \in (1, \ldots, M) \qquad y_i^k \in \{0, 1\} \tag{1.14}$$

Dans ce problème, la contrainte 1.7 vérifie que la capacité de chargement de chaque véhicule est bien respectée. La contrainte 1.8 permet de vérifier le passage unique à chaque sommet du graphe (hors le dépôt) et la contrainte 1.9 permet de vérifier que l'on construit bien M tournées. Les contraintes 1.10 et 1.11 assurent la cohérence entre la visite d'un sommet par un véhicule et le fait que le service est effectué dans ce sommet. Ainsi, pour pouvoir effectuer un service dans un sommet il faut que l'on ait emprunté un arc entrant et un arc sortant du sommet. La contrainte 1.12 est similaire à la contrainte 1.4 pour le TSP et a pour but d'éviter la formation de sous-tours au sein des tournées.

En ce qui concerne la contrainte d'élimination des sous-tours (1.4, 1.12), il est également possible d'utiliser celles proposées par Miller *et al.* (1960) [111] qui ne fait intervenir que n^2 contraintes (au lieu de $2^n - 2$ ici).

1.2.2 Le VRP avec contrainte de fenêtres de visite

Les fenêtres de visite ou fenêtres de temps correspondent à un intervalle de temps pendant lequel une visite chez le client est possible. Deux types de fenêtres de visite existent, une appelée "fenêtre large" et une autre appelée "fenêtre serrée". D'un côté, la fenêtre de visite large autorise un véhicule à arriver en dehors de la fenêtre de visite des clients mais en contre partie une pénalité dépendant de la violation sera alors infligée. De l'autre côté, la fenêtre de visite serrée, n'autorise en aucun cas l'arrivée d'un véhicule en dehors des heures de visite. Le problème de tournées de véhicules avec fenêtres de visite est aussi appelé *Vehicle Routing Problem with Time Windows (VRPTW)*. Pour étendre le modèle proposé par Fisher et Jaikumar (1978, 1981) [64], [65] afin de prendre en considération les fenêtres de visite des différents sommets, introduisons les termes suivants :

- a_i : la borne inférieure de la fenêtre de visite du sommet i ;
- b_i : la borne supérieure de la fenêtre de visite du sommet i ;
- s_i : le temps de service du sommet i ;
- t_{ij} : le temps de transport entre les nœuds i et j ;
- u_i^k : l'instant de visite du sommet i par le véhicule k ;
- T : une grande valeur (T » 0).

Les contraintes de respect des fenêtres de visite peuvent ainsi s'écrire :

$$\forall\, i \in (1, \ldots, n) \quad \forall\, k \in (1, \ldots, M) \qquad a_i \leq u_i^k \leq b_i \tag{1.15}$$

$$\forall\, i \in (1, \ldots, n) \quad \forall\, j \in (2, \ldots, n) \quad \forall\, k \in (1, \ldots, M) \qquad u_i^k + s_i + t_{ij} - T(1 - x_{ij}^k) \leq u_j^k \tag{1.16}$$

La contrainte 1.15 permet de vérifier que l'instant de visite du site se trouve entre les bornes inférieure et supérieure de la fenêtre de visite du site. La contrainte 1.16 vérifie la cohérence des instants de visite lorsque deux sites se suivent.

1.2.3 Les autres extensions

Plusieurs autres extensions sont possibles au problème de *VRP*. Nous pouvons citer notamment : les problèmes sur plusieurs périodes [5] (*PVRP : Periodic Vehicle Routing Problem* ou *PVRPTW : Periodic Vehicle Routing Problem with Time Windows* ou *IRP : Inventory Routing Problem*), les problèmes avec plusieurs dépôts [28] (*MDVRP : multiple depots VRP* ou *MDVRPTW : multiple depots VRPTW*), les problèmes mélangeant les demandes [114], [136] (*VRP with Backhauls* ou *VRP with Pick-Up and Delivery*), les problèmes avec des dépôts intermédiaires pour recharger ou décharger le véhicule [6] (*VRP with Satellite Facilities*), *etc.*

1.2.4 Une classification des problèmes de tournées de véhicules

Desrochers *et al.* (1990) [49] proposent une classification des problèmes de tournées de véhicules. Nous allons pour notre part nous contenter de ne sélectionner que trois critères de classification : la demande, la flotte et les critères d'optimisation. Tout d'abord la demande, elle peut être déterministe ou stochastique. En plus, elle peut être dynamique, c'est à dire qu'elle peut donc changer au cours de la période. Dans les problèmes dynamiques, la résolution se fait souvent en même temps que se fait l'exécution de celui-ci. La demande peut aussi être contrainte à une fenêtre de visite pendant laquelle le sommet doit être visité. De plus la demande peut être contrainte à une précédence, ceci est souvent le cas dans les problèmes de collectes et livraisons. Dans un second temps la flotte, est soit homogène lorsque tous les véhicules qui la composent sont de même capacité, soit hétérogène si ce n'est pas le cas. Enfin, les critères d'optimisation peuvent être divers, retenons-en quelques un : minimiser la distance totale parcourue, minimiser la durée totale du parcours, minimiser la taille de la flotte utilisée, minimiser le coût total du parcours (coût fixe d'utilisation des véhicules plus coût lié au transport), maximiser le nombre de services effectués, maximiser les profits, ...

1.3 Le problème de logistique inverse étudié dans le cadre de cette thèse

Comme évoqué précédemment, la logistique inverse concerne la prise en compte des flux de retour de produits, d'emballages ou de produits de manutention des clients vers le système productif à des fins économiques, environnementales ou de service. Cela concerne les opérations de recyclage, réparation, élimination ...

C'est dans ce contexte que nous nous intéressons à la construction des tournées prenant en compte les flux directs (livraisons) et indirects (collectes) sur un horizon de temps fini.

Nous nous plaçons donc dans un réseau où un produit est livré à partir d'un dépôt vers plusieurs magasins (n magasins) dont les demandes en produits sont connues pour chaque journée. Par ailleurs, les magasins expriment une demande de collecte de produits en retour et de produits de manutention (palettes, bacs, containers...) à destination du dépôt. Chaque magasin i possède une capacité de stockage de type unique qui est de S_i. Chaque magasin, possède une fenêtre de visite. Dans notre étude nous considérons des fenêtres de visite dites "serrées" (*cf.* 1.2.2).

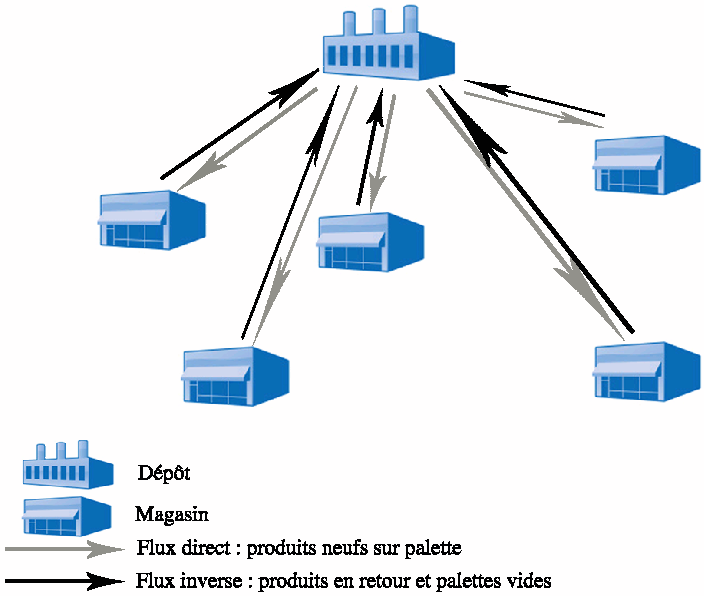
Pour satisfaire les demandes de livraison et de collecte nous disposons sur chaque journée de planification d'une flotte de véhicules homogènes qui partent et reviennent tous au dépôt. La flotte de véhicules est limitée (v véhicules). Sur une même journée, plusieurs véhicules peuvent livrer une même demande, la demande est donc préemptive pour chaque journée. La planification des livraisons s'effectue sur une période de longueur H (exprimée en jours). La demande prévisionnelle du magasin m pour la journée j sera notée D_m^j. Sur chaque palette γ unités de produit sont conditionnées.

Trois différents types de mouvements sont possibles dans ce réseau (Fig.1.2) :

- Les produits sont distribués sur des palettes le jour j. Ils partent du dépôt et vont en direction des magasins ;
- À partir du jour $j+1$ les palettes vides sont retournées des magasins vers le dépôt ;
- Les clients achètent les produits en magasin. Ils ont la possibilité une fois le produit acheté de le retourner au magasin sous α jours s'il ne correspond pas à leur besoin. Lorsqu'un produit est retourné au magasin

par le client, il repart vers le dépôt pour un re-conditionnement. Il pourra repartir en magasin β jours après le retour au dépôt.

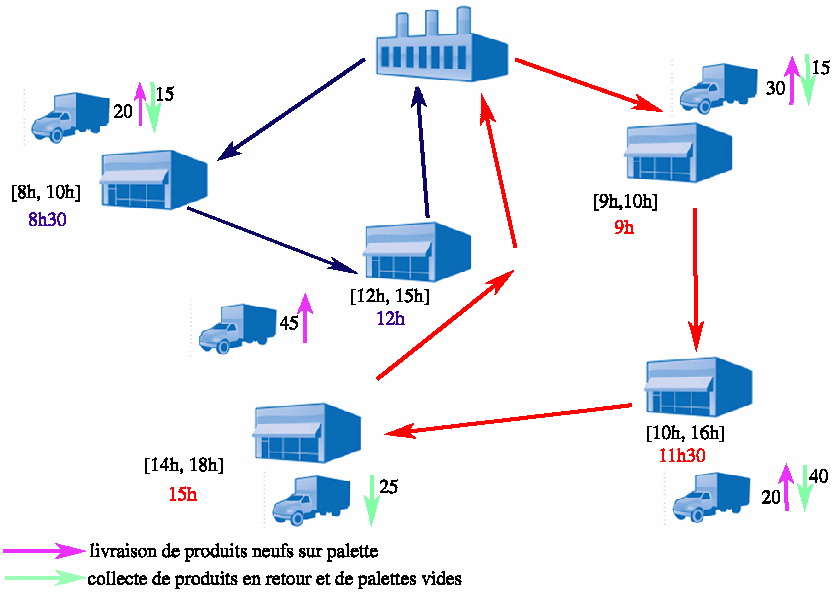
Figure 1.2 *Réseau étudié*



Les caractéristiques des tournées dans ce réseau sont résumées dans la figure 1.3. Les véhicules partent du dépôt chargés du stock de produits qu'ils doivent distribuer dans leur tournée. Le véhicule doit arriver dans la fenêtre de visite pour chaque site visité. Chacun des sites est soit livré en produits soit collecté en produits en retour et en palettes vides soit les deux en même temps. Ce qui amène le véhicule à retourner au dépôt également chargé des diverses collectes.

Le réseau présenté dans la figure 1.2 est inspiré du fonctionnement des grandes enseignes de supermarchés de

Figure 1.3 *Caractéristiques des tournées dans le réseau*



type Système U.

Dans la suite des chapitres nous allons considérer deux stratégies de livraison :

– *En juste à temps* : consiste à répondre à la demande des sites le jour donné (*ex.* : si le site a besoin de 20 produits le jour 2 alors on effectuera une livraison de 20 produits le jour 2) ;

– *Avec gestion des stocks* : on s'autorise à livrer par avance certaines demandes, un coût de stockage est donc à payer pour toute demande livrée avec avance. Il est peut-être plus avantageux de payer un coût de stockage que de payer le coût de routage de la demande le jour voulu.

1.3.1 Une classification du problème

Notre problème correspond à un problème de collectes et livraisons de produits, avec ou sans gestion des stocks des magasins. Il peut donc se classer dans la catégorie IPDPTW (Inventory Pick-up and Delivery Problem with Time Windows) comme a pu l'introduire Christiansen (1999) [30]. Puisque nous autorisons le partage de la demande nous pouvons préciser que notre problème appartient à la catégorie : *SIPDPTW* (*Split Inventory Pick-up and Delivery Problem with Time Windows*). Par conséquent nous pouvons dire que notre problème est donc *NP-difficile* puisqu'il est une extension du problème du PDPTW.

1.3.2 La division des demandes

Dans notre réseau, nous avons choisi de pouvoir desservir une même demande de livraison grâce à plusieurs véhicules. Comme le démontre Gendreau *et al.* (2006) [70], il est parfois plus économique de considérer une demande comme préemptive. Pour illustrer cela, nous allons reprendre l'exemple que Gendreau *et al.* utilisent.

Exemple *Considérons un cercle de rayon M et dont le centre est le dépôt, supposons que nous avons n clients avec $n = 2k$ qui sont localisés à équidistance sur le cercle. La demande de chaque client est de k et la capacité des véhicules est de $Q = 2k - 1$. La distance entre chaque client est de ϵ. La solution optimale sans partager la demande est de visiter chaque site indépendamment, on obtient ainsi un coût de $2nM$. Alors, que lorsque le partage des demandes est autorisé une solution consiste à visiter deux clients consécutifs, en ne livrant que $k-1$ au deuxième client (soit sa demande moins une unité), la dernière tournée livrera alors les k clients auxquels il manque une unité de leur demande. Chacune des k premières tournées a un coût de $2M+\epsilon$ et la dernière tournée un coût de $2M + 2\pi M$. Ce qui au total donne un coût de $k(2M+\epsilon)+2M+2\pi M$ c.-à-d. $nM(1+\frac{\epsilon}{2M}+\frac{2+2\pi}{n})$. Ce qui démontre que le ratio entre la valeur optimale obtenue pour le problème avec partage de la demande et la valeur optimale obtenue sans tend vers $\frac{1}{2}$ lorsque n tend vers ∞. De même on peut noter que le nombre de véhicules utilisés est diminué.*

Gendreau *et al.* (2006) [70] se basent sur les travaux de Dror et Trudeau (1990) [55] pour démontrer que dans une solution optimale du *SDVRPTW* (*Split Delivery Vehicle Routing Problem with Time Windows*) il ne peut pas y avoir deux tournées ayant plus d'une demande de livraison en partage en commun. Pour prouver ceci ils utilisent la preuve suivante :

Considérons une solution optimale pour le SDVRPTW dans laquelle nous avons les deux tournées suivantes : r_k et r_l. Ces deux tournées ont deux sites en commun v_i et v_j. q_i^k, q_i^l, q_j^k et q_j^l sont les quantités positives que les tournées r_k et r_l livrent au site v_i et v_j. Notons que $q_j^l = min(q_i^k, q_i^l, q_j^k, q_j^l)$. Nous pouvons modifier les tournées de façon à ce que les quantités livrés à v_i et v_j deviennent : $q_i^{'k} = q_i^k - q_j^l$, $q_i^{'l} = q_i^l + q_j^l$, $q_j^{'k} = q_j^k + q_j^l$ et $q_j^{'l} = 0$. Dans cette nouvelle solution la quantité livrée aux deux sites sont identiques, la charge total dans le véhicule est la même, les fenêtres de visite sont toujours respectées si le temps de service ne dépend pas de la quantité livrée. Enfin, si les temps de transport et les coûts de transport respectent l'inégalité triangulaire, le coût de la deuxième solution est meilleur que celui de la première ce qui prouve qu'aucune solution optimale ne peut contenir deux tournées contenant plus d'un site de livraison en commun.

Ce théorème ne peut pas s'appliquer dans notre cas d'étude. En effet, la présence de points de collecte pouvant être possible à n'importe quel endroit dans une tournée, la preuve énoncée ci-dessus ne peut pas être validée dans tous les cas. Prenons par exemple le cas suivant : les véhicules ont une capacité de 100, la tournée 1 visite les sites 2, 3 et 4 et la tournée 2 visite les sites 1, 2 et 4. La tournée 1 effectue les actions suivantes : livraison de 35 en 2, collecte de 60 en 3 et livraison de 40 en 4. La tournée 2 effectue les actions suivantes : livraison de 20 en 1, livraison de 40 en 2 et livraison de 40 en 4. Les deux tournées passent toutes les deux par les sites 2 et 4. Dans notre cas q_j^l = 35, en suivant la logique de la démonstration on arrive à une tournée $1^{'}$: livraison

de 0 à 2, collecte de 60 à 3 et livraison de 75 à 4. Or pour livrer les 75 à 4 il faut les avoir dans le véhicule dès le départ il est donc impossible de collecter les 60 en 3 sans violer la contrainte de respect de capacité du véhicule.

1.3.3 Le modèle mathématique pour la version en juste à temps

Dans cette partie nous allons proposer le modèle mathématique pour notre problème dans sa version en juste à temps.

Les indices

- I : nombre de magasins ;
- T : nombre de périodes ;
- V : nombre de véhicules ;
- i (ou i') : indice des magasins, i = 0 représente le dépôt et $i \in \{1, \ldots, I\}$ les magasins ;
- t : indice des périodes, $t \in \{1, \ldots, T\}$;
- v : indice des véhicules, $v \in \{1, \ldots, V\}$.

Les magasins (i > 0)

- Sm_i : capacité totale de stockage (en m^3) du magasin i ;
- D_i^t : demande prévisionnelle du magasin i pour la journée t (en palettes pleines) ;
- V_i^t : ventes réelles du magasin i le jour t (en palettes pleines) ;
- $Smin_i$: niveau de stock minimal de palettes pleines que doit avoir le magasin i (en palettes pleines) ;
- r_i^t : nombre de palettes arrivant en retour du client au magasin i le jour t (en palettes pleines) ;
- $d_{ii'}$: distance séparant le magasin i du magasin i' (en km) ;
- a_i : borne inférieure de la fenêtre de visite du magasin i (horaire) ;
- b_i : borne supérieure de la fenêtre de visite du magasin i (horaire) ;
- s_i : temps de service du site i (en minutes).

Le dépôt (i = 0)

- d_{0i} : distance séparant le dépôt du magasin i (en km).

Flotte de véhicules

- Sv_v : capacité de transport du véhicule v (en m^3) ;
- k : représente la vitesse moyenne des véhicules (en km/h).

Les coûts

Les différents coûts sont exprimés en unité monétaire. Tout d'abord voyons les coûts liés aux véhicules :
- Cf_v : coût fixe d'utilisation du véhicule v (en euro) ;
- Cv_v : coût variable d'utilisation du véhicule v (euro/km) ;

Maintenant voyons les coûts liés aux stocks :
- CS : coût de stockage par jour d'une palette pleine (en euro) ;

Volume des différents éléments

- a : volume d'une palette vide (en m^3) ;
- b : volume d'une palette pleine (en m^3) ;
- c : volume d'un produit en retour (en m^3) ;
- $txPal$: taux de palettisation *c.-à-d.* nombre de produits sur une palette ;
- $txPalV$: nombre de palettes vides correspondant au volume d'une palette pleine.

Variables de décision

Pour les magasins :

- XNm_i^t : niveau de stock en palettes pleines dans le magasin i le jour t ;
- $XPal_i^t$: niveau de stock de palettes vides au magasin i le jour t ;
- $XPdt_i^t$: niveau de stock de produits à retourner du magasin i vers le dépôt le jour t ;
- t_{iv} : instant de visite du site i par le véhicule v.

Pour les véhicules :

- XNv_{vi}^t : niveau de stock (en m^3) dans le véhicule v à son arrivée au site i le jour t ;
- XI_v^t : nombre de sites visités par le véhicule v le jour t ;
- $x_{ii'v}^t$: variable binaire égale à 1 si le véhicule v visite le magasin i' immédiatement après le magasin i le jour t ; sinon 0 ;
- y_v^t : variable binaire égale à 1 si le véhicule v est utilisé le jour t ; sinon 0 ;
- y_{iv}^t : variable binaire égale à 1 si le véhicule v est utilisé le jour t pour visiter le site i ; sinon 0 ;
- $XAppro_{iv}^t$: nombre de palettes pleines livrées au magasin i par le véhicule v le jour t ;
- z_{iv}^t : nombre de palettes vides prises par le véhicule v au magasin i le jour t à destination du dépôt ;
- w_{iv}^t : nombre de produits en retour repris par le véhicule v au magasin i le jour t à destination du dépôt.

L'objectif

Le but est de minimiser les coûts totaux (c.-à-d. les coûts de routage).

$$\min f = \sum_{v=1}^{V}\sum_{t=1}^{T} y_v^t Cf_v + \sum_{i=0}^{I}\sum_{\substack{i'=0 \\ i'\neq i}}^{I}\sum_{v=1}^{V}\sum_{t=1}^{T} x_{ii'v}^t Cv_v d_{ii'} \tag{1.17}$$

Le coût à minimiser prend en compte le fait qu'à chaque utilisation d'un véhicule, on ajoute son coût fixe d'utilisation et on ajoute également les frais d'utilisation kilométrique.

Modèle

$$\forall\, i > 0 \quad \forall\, t \qquad aXPal_i^t + cXPdt_i^t + bXNm_i^t \leq Sm_i \tag{1.18}$$

$$\forall\, i > 0 \quad \forall\, t \qquad XNm_i^{t+1} = XNm_i^t - V_i^t + \sum_{v=1}^{V} XAppro_{iv}^t \tag{1.19}$$

$$\forall\, v \quad \forall\, t \quad \forall\, i \qquad Sv_v \geq XNv_{vi}^t - b \times XAppro_{iv}^t + c \times w_{iv}^t + a \times z_{iv}^t \tag{1.20}$$

$$\forall\, t \quad \forall\, v \quad M > 0 \qquad y_v^t \leq \sum_{i=0}^{I} \sum_{\substack{i'=0 \\ i' \neq i}}^{I} x_{ii'v}^t \leq M y_v^t \tag{1.21}$$

$$\forall\, i > 0 \quad \forall\, t \qquad XPal_i^{t+1} = XPal_i^t - \sum_{v=1}^{V} z_{iv}^t + \frac{V_i^t}{txPal} \tag{1.22}$$

$$\forall\, i > 0 \quad \forall\, t \qquad XPdt_i^{t+1} = XPdt_i^t - \sum_{v=1}^{V} w_{iv}^t + r_i^t \tag{1.23}$$

$$\forall\, t \quad \forall\, v \quad \forall\, i \quad M >> 0 \qquad \sum_{\substack{i'=0 \\ i' \neq i}}^{I} x_{ii'v}^t \leq (z_{iv}^t + w_{iv}^t + XAppro_{iv}^t) \leq M \sum_{\substack{i'=0 \\ i' \neq i}}^{I} x_{ii'v}^t \tag{1.24}$$

$$\forall\, i \quad \forall\, t \quad \forall\, v \qquad \sum_{\substack{i'=0 \\ i' \neq i}}^{I} x_{ii'v}^t = y_{iv}^t \tag{1.25}$$

$$\forall\, i \quad \forall\, t \quad \forall\, v \qquad \sum_{\substack{i'=0 \\ i' \neq i}}^{I} x_{i'iv}^t = y_{iv}^t \tag{1.26}$$

$$\forall\, t \quad \forall\, v \qquad \sum_{i=1}^{I} \sum_{\substack{i'=1 \\ i' \neq i}}^{I} x_{ii'v}^t \leq XI_v^t - 1 \tag{1.27}$$

$$\forall\, t \qquad \sum_{v=0}^{V} \sum_{i'=1} x_{0i'v}^t \leq V \tag{1.28}$$

$$\forall\, t \qquad \sum_{v=0}^{V} \sum_{i'=1} x_{i'0v}^t \leq V \tag{1.29}$$

$$\forall\, i \quad \forall\, v \qquad a_i \leq t_{iv} \leq b_i \tag{1.30}$$

$$\forall\, i \quad \forall\, j \quad \forall\, v\, \forall\, t \qquad t_{iv} + s_i + d_{ij} - T(1 - x_{ijv}^t) \leq t_{jk} \tag{1.31}$$

$$\forall\, i \quad \forall\, t \qquad \sum_{v=1}^{V} XAppro_{iv}^t = D_i^t \tag{1.32}$$

$$\forall\, v \quad \forall\, t \qquad y_v^t \in [0, 1] \tag{1.33}$$

$$\forall\, i \quad \forall\, i' \quad \forall\, v \quad \forall\, t \qquad y_{iv}^t,\ x_{ii'v}^t \in [0, 1] \tag{1.34}$$

$$\forall\, i \quad \forall\, i' \quad \forall\, v \quad \forall\, t \qquad \beta_{ii'v}^t \in \mathbb{N} \tag{1.35}$$

$$\forall\, v \quad \forall\, i \qquad w_{iv}^t \in \mathbb{N} \tag{1.36}$$

$$\forall\, v \quad \forall\, i \qquad z_{iv}^t \in \mathbb{N} \tag{1.37}$$

Le tableau 1.1 explique chaque contrainte de la version en juste à temps du problème.

(1.18)	Dans chacun des magasins, chaque jour, le volume de palettes vides plus le volume de produits à retourner, plus le volume de palettes pleines ne doit pas dépasser la capacité (en volume) de stockage du magasin.
(1.19)	Le stock de palettes pleines au magasin à la période $t+1$ est égal au stock de palettes pleines dans ce magasin à la période t moins les ventes qui ont été effectuées à la période t, plus l'éventuel réapprovisionnement de palettes qui aurait été fait à la période t.
(1.20)	Pour chaque site visité par le véhicule v, il faut que le niveau de stock dans le véhicule à son arrivée au site moins les palettes pleines descendues à ce site, plus les palettes vides reprises, ne dépasse pas la capacité de transport du véhicule v.
(1.21)	Lorsque deux points sont reliés par un véhicule, le véhicule est utilisé.
(1.22)	Le stock de palettes vides pour le magasin i à la période $t+1$ est égal au stock de palettes vides de la période t pour ce magasin moins les palettes vides qui ont été récupérées par les différents véhicules, plus les palettes vides provenant de la vente des produits.
(1.23)	Idem que pour l'équation (1.22) mais pour les palettes de retour.
(1.24)	Un véhicule v ne peut pas prendre de palettes vides dans un magasin à la période t s'il ne passe pas par ce magasin sur cette même période. Il en est de même avec les retours de produits. De plus, un véhicule v ne peut pas livrer de palettes provenant du dépôt au magasin i à la période t s'il ne passe pas par ce magasin à cette même période. Inversement, un véhicule v ne doit pas passer par le magasin i le jour t s'il n'a rien à faire au magasin i (pas de dépôt, pas de reprise de palettes vides ni de produits en retour).
(1.25) (1.26)	Assurent qu'au plus un seul site sera visité immédiatement après (avant) le site i par le véhicule v le jour t si et seulement si i est visité par ce même véhicule le même jour.
(1.27)	Contrainte interdisant les sous tours.
(1.28) (1.29)	Assurent qu'il y a au maximum V tournées sur chaque journée de la planification.
(1.30) (1.31)	Assurent le respect des fenêtres de visite des sites.
(1.32)	Assure le juste à temps de la livraison de la demande : on ne livre que ce qu'a besoin le site le jour considéré

TAB. 1.1 – Explications des contraintes dans la version en juste à temps

1.3.4 Le modèle mathématique pour la version avec gestion des stocks

L'objectif

Le but est de minimiser les coûts totaux (*c.-à-d.* les coûts de stockage plus les coûts de livraisons).

$$\min f = (\sum_{v=1}^{V}\sum_{t=1}^{T} y_v^t Cf_v + \sum_{i=0}^{I}\sum_{\substack{i'=0\\i'\neq i}}^{n}\sum_{v=1}^{V}\sum_{t=1}^{T} x_{ii'v}^t Cv_v d_{ii'}) \tag{1.38}$$
$$+CS(\sum_{i=0}^{I}\sum_{t=0}^{T}(\sum_{v=0}^{V} XAppro_{iv}^t) - D_i^t)$$

Dans la première partie, à chaque utilisation d'un véhicule on ajoute son coût fixe d'utilisation et on ajoute également les frais d'utilisation kilométrique.

Dans la seconde partie, nous prenons en compte les coûts de stockage en magasin.

Modèle

$$\forall\, i>0\,\forall\, t \qquad aXPal_i^t + cXPdt_i^t + bXNm_i^t \leq Sm_i \tag{1.39}$$

$$\forall\, i>0\,\forall\, t \qquad XNm_i^{t+1} = XNm_i^t - V_i^t + \sum_{v=1}^{V} XAppro_{iv}^t \tag{1.40}$$

$$\forall\, v\,\forall\, t\,\forall\, i \qquad Sv_v \geq XNv_{vi}^t - b\times XAppro_{iv}^t + c\times w_{iv}^t + a\times z_{iv}^t \tag{1.41}$$

$$\forall\, t\,\forall\, v\; M>0 \qquad y_v^t \leq \sum_{i=0}^{I}\sum_{\substack{i'=0\\i'\neq i}}^{I} x_{ii'v}^t \leq My_v^t \tag{1.42}$$

$$\forall\, i>0\,\forall\, t \qquad XPal_i^{t+1} = XPal_i^t - \sum_{v=1}^{V} z_{iv}^t + \frac{V_i^t}{txPal} \tag{1.43}$$

$$\forall\, i>0\,\forall\, t \qquad XPdt_i^{t+1} = XPdt_i^t - \sum_{v=1}^{V} w_{iv}^t + r_i^t \tag{1.44}$$

$$\forall\, t\,\forall\, v\,\forall\, i\; M>>0 \qquad \sum_{\substack{i'\neq 0\\i'\neq i}}^{I} x_{ii'v}^t \leq (z_{iv}^t + w_{iv}^t + XAppro_{iv}^t) \leq M\sum_{\substack{i'\neq 0\\i'\neq i}}^{I} x_{ii'v}^t \tag{1.45}$$

$$\forall\, i\,\forall\, t\,\forall\, v \qquad \sum_{\substack{i'=0\\i'\neq i}}^{I} x_{ii'v}^t = y_{iv}^t \tag{1.46}$$

$$\forall\, i\,\forall\, t\,\forall\, v \qquad \sum_{\substack{i'=0\\i'\neq i}}^{I} x_{i'iv}^t = y_{iv}^t \tag{1.47}$$

$$\forall\, t\,\forall\, v \qquad \sum_{i=1}^{I}\sum_{\substack{i'=1\\i'\neq i}}^{I} x_{ii'v}^t \leq XI_v^t - 1 \tag{1.48}$$

$$\forall\, t \qquad \sum_{v=0}^{V} \sum_{i'=1} x^t_{0i'v} \leq V \tag{1.49}$$

$$\forall\, t \qquad \sum_{v=0}^{V} \sum_{i'=1} x^t_{i'0v} \leq V \tag{1.50}$$

$$\forall\, t\, \forall\, v\, \forall i \in XI^t_v \qquad e_i \leq T^t_{iv} \leq l_i \tag{1.51}$$

$$\forall\, t\, \forall\, v\, \forall\, i\, \forall\, i' \quad M \gg 0 \quad (1 - x^t_{ii'v})M \geq T^t_{iv} + s_i + kd_{ii'} - T^t_{i'v} \tag{1.52}$$

$$\forall\, i \quad \forall\, t \qquad XNm^t_i \geq D^t_i + Smin_i \tag{1.53}$$

$$\forall\, v\, \forall\, t \qquad y^t_v \in [0,1] \tag{1.54}$$

$$\forall\, i\, \forall\, i'\, \forall\, v\, \forall\, t \qquad y^t_{iv},\ x^t_{ii'v} \in [0,1] \tag{1.55}$$

$$\forall\, i\, \forall\, i'\, \forall\, v\, \forall\, t \qquad \beta^t_{ii'v} \in \mathbb{N} \tag{1.56}$$

$$\forall\, v\, \forall\, i \qquad w^t_{iv} \in \mathbb{N} \tag{1.57}$$

$$\forall\, v\, \forall\, i \qquad z^t_{iv} \in \mathbb{N} \tag{1.58}$$

Le tableau 1.2 explique les contraintes présentent dans le problème avec gestion des stocks.

1.4 Conclusion

Dans ce chapitre nous avons défini dans quel contexte notre travail s'inscrit. Nous avons ainsi décrit la problématique de la logistique inverse en utilisant les différentes définitions que nous avons trouvées. Nous avons également décrit les différents traitements des retours dans la logistique inverse et les intérêts à mettre en place un tel réseau. Nous avons ensuite rappelé les définitions des problèmes de construction de tournées de véhicules et donné quelques explications concernant l'option de pouvoir livrer une demande grâce à plusieurs véhicules. Enfin nous avons défini notre problème et donné les deux modèles mathématiques qui le caractérisent.

Dans la suite de ce document nous allons voir comment résoudre ce problème de construction de tournées dans le cadre de la logistique inverse. Nous nous attacherons à distinguer les deux stratégies énoncées auparavant (juste à temps et avec gestion des stocks).

(1.39)	Dans chacun des magasins, chaque jour, le volume de palettes vides plus le volume de produits à retourner, plus le volume de palettes pleines ne doit pas dépasser la capacité (en volume) de stockage du magasin.
(1.40)	Le stock de palettes pleines au magasin à la période $t+1$ est égal au stock de palettes pleines dans ce magasin à la période t moins les ventes qui ont été effectuées à la période t, plus l'éventuel réapprovisionnement de palettes qui aurait été fait à la période t.
(1.41)	Pour chaque site visité par le véhicule v, il faut que le niveau de stocks dans le véhicule à son arrivée au site moins les palettes pleines descendues à ce site (venue du dépôt), plus les palettes vides reprises, ne dépassent pas la capacité de transport du véhicule v.
(1.42)	Lorsque deux points sont reliés par un véhicule, le véhicule est utilisé.
(1.43)	Le stock de palettes vides pour le magasin i à la période $t+1$ est égal au stock de palettes vides de la période t pour ce magasin moins les palettes vides qui ont été récupérées par les différents véhicules, plus les palettes vides provenant de la vente des produits.
(1.44)	Idem que pour l'équation (1.43) mais pour les palettes de retour.
(1.45)	Un véhicule v ne peut pas prendre de palettes vides dans un magasin à la période t s'il ne passe pas par ce magasin sur cette même période. Il en est de même avec les retours de produits. De plus, un véhicule v ne peut pas livrer de palettes provenant du dépôt au magasin i à la période t s'il ne passe pas par ce magasin à cette même période. Inversement, un véhicule v ne doit pas passer par le magasin i le jour t s'il n'a rien à faire au magasin i (pas de dépôt, pas de reprise de palettes vides ni de produits en retour).
(1.46) (1.47)	Assurent qu'au plus, un seul magasin ne sera visité immédiatement après (avant) le magasin i par le véhicule v le jour t si et seulement si i est visité par ce même véhicule le même jour.
(1.48)	Contrainte interdisant les sous tours.
(1.49) (1.50)	Assurent qu'il y a au maximum V tournées sur chaque journée de la planification.
(1.51)	Assure le respect des fenêtres de visite des magasins.
(1.52)	Le successeur du magasin i dans la tournée réalisée par le véhicule v le jour t aura un temps de visite supérieur à celui de i plus son temps de service plus le temps de parcours de la distance séparant les deux magasins.
(1.53)	Assure le fait qu'il y a assez pour satisfaire la demande du jour.

TAB. 1.2 – Explications des contraintes pour le modèle avec gestion des stocks

Chapitre 2

État de l'art

Nous allons dans ce chapitre voir un échantillon des travaux qui ont pu être réalisés dans les différents domaines qui ont trait à notre problème. Ainsi, nous verrons tout d'abord les travaux qui se rapportent au contexte de notre étude : la logistique inverse. Puis nous verrons les travaux sur les tournées de véhicules qui correspondent à la base du problème qui nous intéresse. Les méthodes hybrides, fruits de l'utilisation de méthodes issues de la recherche opérationnelle avec des techniques issues de la programmation par contraintes,[1] dont l'efficacité a été prouvée sur les problèmes de transport, font l'objet de la troisième partie de ce chapitre.

2.1 La logistique inverse

Comme nous avons pu le définir dans le chapitre précédent, la logistique inverse considère les flux dits inverses (allant du consommateur vers le producteur), en opposition aux flux dits traditionnels (allant du producteur vers les consommateurs). Nous classerons les différents documents que nous avons trouvés dans la littérature[2] selon les trois niveaux de planification hiérarchisée, qui sert de structure de modélisation. Elle s'appuie sur une structure décisionnelle échelonnée du haut vers le bas, dans lequel un problème global est décomposé en une série de sous-problèmes qui correspondent aux différents niveaux décisionnels et dont la solution est obtenue lorsque l'ensemble des sous-problèmes est résolu.

Avant de poursuivre nous allons rappeler ici les définitions des trois niveaux de la planification hiérarchisée :

Le niveau stratégique : *les décisions de la planification stratégique sont les grandes orientations de l'entreprise, les décisions prises à haut niveau, l'investissement important, à long terme. Comme exemple de décisions nous pouvons citer : la localisation et la taille de nouvelles usines, l'acquisition de nouveaux équipements, etc.*

Le niveau tactique : *les décisions de la planification tactique sont des problèmes d'affectation de ressources : main d'œuvre, capacité, ressources d'entreposage, gestion des stocks, etc. Le but étant de satisfaire la demande de la manière la plus rentable et efficace possible.*

Le niveau opérationnel : *les décisions sur ce niveau sont des problèmes opérationnels : construction de tournées, planification de la production, etc. et d'ordonnancement quotidien.*

Nous passerons plus rapidement sur les deux premiers niveaux (stratégique et tactique) puisque nos travaux se concentrent sur le niveau opérationnel de la planification hiérarchique.

2.1.1 Au niveau stratégique

Les travaux menés sur ce niveau de planification sont des études essentiellement de mise en place d'un réseau de logistique inverse : organisation et élaboration du réseau. Les décisions qui vont être prises suite à ces études sont de grandes décisions stratégiques qui reflètent la politique de l'entreprise.

[1] Des rappels des techniques de base sur la programmation par contraintes et la programmation linéaire sont faits respectivement dans les annexes 1 et 2. Ces deux types de méthodes de résolution sont complémentaires et permettent de palier aux lacunes de l'une ou de l'autre.

[2] Notons qu'un état de l'art complet sur la logistique inverse a été réalisé par Bostel *et al.* (2005)[20], nous invitons le lecteur à s'y référer pour de plus amples renseignements.

Les différents objectifs possibles pour les études sur le niveau stratégique sont : la détermination des acteurs d'un réseau et de leur rôle (Beaulieu *et al.*, 1999 [10]), le choix du réseau *"open-loop"*[3] ou *"close-loop"*[4] (Fleischmann *et al.*, 1997 [67]), le choix du type de reconversion qu'il faut choisir pour les produits en retour (Thierry *et al.*, 1993 [149]), *etc.*

Dans les différentes études les modèles peuvent traiter indépendamment les flux inverses du réseau logistique classique (Barros *et al.*, 1998 [8]), les modèles peuvent traiter les flux directs et inverses simultanément avec une faible corrélation entre les deux flux (Lu *et al.*, 2004 [107]) et enfin les modèles peuvent traiter les deux flux ensembles avec une forte corrélation entre les deux flux du réseau (Lu, 2003 [106]).

Par exemple, Bloemhof-Ruwaard *et al.* (1994) [18] font une étude sur le problème de la conception d'un réseau de distribution (avec comme acteurs : des usines, des clients et des centres de traitements des déchets) et sur le problème de la coordination des flux de produits (flux directs) et de déchets (flux inverses) dans ce réseau. Ce problème est appliqué à un problème de coordination de flux entre des entreprises agricoles, des entreprises agro-alimentaires et des usines de traitement des engrais.

C'est dans le domaine du recyclage des déchets aciers dus aux démolitions de bâtiments que Spengler *et al.* (1997) [142] réalisent leur étude. L'originalité de ces travaux réside dans le fait que les flux inverses constituent ici l'étape avant production. Le but étant de déterminer quel procédé de recyclage doit être développé pour chacun des produits résiduels, la capacité de recyclage requise, la localisation des installations et l'assignation des produits résiduels vers un site. Un gain de 40% a été fait grâce à la mise en place de ce modèle.

Enfin l'étude menée par Barros *et al.* (1998) [8], est réalisée sur un problème de recyclage de sable de construction. Le but étant ici de déterminer le nombre et le type d'installations qui doivent être implantées en vue de la mise en place du recyclage. De plus il faudra également déterminer où implanter ces installations et leur capacité.

2.1.2 Au niveau tactique

Passons maintenant aux études réalisées sur le deuxième niveau de la planification hiérarchisée : le niveau tactique. Nous allons dans un premier temps voir les différentes études effectuées sur les modèles de gestion de stocks. Puis nous verrons quelques exemples de cas réels que nous avons rencontrés.

Dans les différents travaux réalisés sur la gestion des stocks dans la logistique inverse, deux types de modèle ont été observés.

Dans un premier temps, nous avons un modèle stochastique où les demandes et les retours sont connus à une certaine probabilité près. Dans ce cas d'étude, nous pouvons citer les travaux de Kiesmüller (2003) [96], Kiesmüller et Scherer (2003) [97], Fleischmann *et al.* (2002) [68], Van der Laan et Salomon (1997) [158] ou encore Inderfurth (1997) [90].

Dans ces études stochastiques trois optiques ont été observées :
- une étude sur une seule période (Vlachos et Dekker (2003) [161]) ;
- une étude sur plusieurs périodes (Inderfurth (1997) [90], Kiesmüller (2003) [96], Kiesmüller et Scherer (2003) [97]) ;
- une étude en continu (Fleischmann *et al.* (2002) [68],Van der Laan et Salomon (1997) [158]).

Dans un second temps nous avons un modèle déterministe où les demandes et retours de produits sont connus de façon sûre. Ici, notons les études de : Teunter (2001) [147], Dobos (2003) [51] ou encore Minner et Kleber (2001) [112], *etc.*

Les variantes des problèmes résident sur :
- le type de demande étudié est : déterministe (Beltran et Krass, 2002 [13]) ou stochastique (Van der Laan & Salomon, 1997 [158]) ;
- le nombre de points de stockage : un (Fleischmann *et al.*, 2002 [68]) ou deux (Inderfurth, 1997 [90]) ;
- la prise en compte des coûts fixes (Dobos, 2003 [51]) ou non (Minner et Klerber, 2001 [112]) ;
- la présence de temps d'attente avant l'obtention de la commande ou des retours (Kiesmüller, 2003 [96]) ou non (Van der Laan et Salomon, 1997 [158]) ;

[3]Circuit ouvert : les flux directs partent d'un endroit qui n'est pas le lieu d'arrivée des flux inverses

[4]Circuit fermé : les flux directs partent du lieu où arrivent les flux inverses

- la possibilité de jeter les produits en retour (Teunter, 2001 [147]) ou non (Fleischmann *et al.*, 2002 [68]) ;
- la possibilité d'avoir des commandes en attente (Inderfurth, 1997 [90]) ou non (Kiesmüller, 2003 [96]) ;
- les méthodes utilisées pour trouver les quantités de commande : modèle EOQ[5] comme dans Ritcher (1996) [129], Wagner-Within dans Beltran et Krass (2002) [13] et heuristiques comme dans Kiesmüller et Scherer (2003) [97].

Voyons quelques exemples de cas réels que nous avons rencontrés.

Nous pouvons dans un premier temps noter l'étude de Rudi *et al.* (2000) [134] qui est réalisée sur les retours de produits médicaux (comme des fauteuils roulants, des appareils auditifs) pour Norwegian National Insurance Administration. Dans cette étude, les retours de matériels médicaux sont étudiés pour pouvoir choisir entre la réutilisation ou l'abandon de ceux-ci. Ainsi, ils développent un modèle de système d'aide à la décision qui fournit le coût d'envoi d'une unité à l'enfouissement des déchets, le coût de remise à neuf, la valeur des parties qui peuvent être utilisées ailleurs, les avantages de remise à neuf.

Notons aussi, les retours de matériels informatiques tels que les ordinateurs pour l'entreprise *IBM* qui ont pu être étudiés par Fleishmann (2001) [66]. Dans cette étude, l'auteur aborde le problème selon deux approches. La première consiste à ajouter la logistique inverse à un réseau de logistique existant alors que la deuxième consiste à faire un nouveau réseau logistique alliant les deux types de flux (direct et inverse).

Dans les travaux de Toktay *et al.* (2000) [151], il s'agit des retours d'appareils photos jetables pour l'entreprise Kodak qui sont étudiés. Ainsi, l'étude consiste à développer et à analyser un modèle de la chaîne d'approvisionnement.

Enfin, Van Der Laan (1997) [157] étudie le fait que dans l'industrie automobile certaines pièces rénovées (démarreur, alternateur, *etc.*) sont vendues moins chers que les pièces neuves.

2.1.3 Au niveau opérationnel

Passons maintenant au troisième niveau de la planification hiérarchisée : le niveau opérationnel. Il s'agit du niveau sur lequel se situe le problème étudié dans le cadre de cette thèse. Nous verrons dans un premier temps les différents cas réels étudiés, puis nous classerons les études selon le type de problème traité, le type de flux pris en compte, le nombre de périodes étudiées et la méthode de résolution utilisée.

Les cas réels

Trois études peuvent être considérées : les flux de production, les flux de distribution ou les deux flux traités simultanément.

Tout d'abord commençons par la gestion des flux de production. Krikke *et al.* (1999) [101], réalisent une étude sur le recyclage d'écrans d'ordinateurs pour l'entreprise Roteb. Le but ici est d'analyser la viabilité économique du recyclage de moniteurs et de valider la viabilité pratique des modèles. Ainsi les auteurs ont mis au point un modèle pour le recyclage de moniteurs et ont comparé la solution optimale écologique à celle économique. Les résultats montrent qu'un gain de 25% peut être réalisé sur les coûts de recyclage. De même, l'étude menée par Spengler (2003) [141] a été réalisée pour le compte d'une société allemande de recyclage de déchets électroniques. Il consiste en la conception et la mise en œuvre d'un système d'aide à la décision pour des sociétés de recyclage de déchets électroniques dans des chaînes d'approvisionnement en boucle fermée.

Dans le cas des différents travaux qui portent sur l'optimisation des flux de distribution dans la logistique inverse, notons tout d'abord l'étude réalisée par Crainic *et al.* (1993) [39] dans le domaine du transport de marchandises en containers. Les auteurs proposent un modèle multi-périodique et stochastique pour l'assignation des containers vides. Ce modèle est destiné au transport terrestre de containers maritimes pour le commerce international. De même nous pouvons citer l'étude de Duhaime *et al.* (2001) [56] sur la réutilisation des containers de Canada Post. Ils montrent grâce à un modèle de flux de coût minimal que les ruptures de stocks peuvent être évitées si les containers sont retournés rapidement. Citons les travaux de Feillet *et al.* (2002) [59] qui sont réalisés dans l'industrie automobile pour un transport inter-usine de produits en container. Le but est ici d'optimiser les circuits inter-usines combinant le transport de containers chargés, les retours de containers vides et le positionnement des camions vides.

Finalement voyons l'étude de Del Castillo et Cochran (1996) [43], où les deux types de flux (production

[5] Economic Order Quantity

et distribution) sont traités pour des containers réutilisables. En effet, cette étude porte sur la production et la distribution de produits livrés dans des containers réutilisables. Ainsi, les retours des containers vides sont une contrainte pour la production.

Les types de problème :

Plusieurs types de problème ont été étudiés. En effet, nous avons pu voir des études sur la distribution de produits comme dans Crainic *et al.* (1993) [39], Del Castillo et Cochran (1996) [43], Duhaime *et al.* (2001) [56], Feillet *et al.* (2002) [59]. Nous avons également pu voir des études sur la production de produits comme dans Krikke *et al.* (1998) [100], Spengler *et al.* (1997) [142], Gupta et Taleb (1994) [82], Thierry (1997) [148]. Et enfin nous avons pu voir des études sur la production et distribution de produits comme dans Del Castillo et Cochran (1996) [43], Lu (2003) [106].

Le type de flux pris en compte :

En ce qui concerne les flux qui sont pris en compte dans les études sur le niveau opérationnel de la logistique inverse, nous avons constaté deux orientations possibles. Une première orientation est de ne considérer qu'une étude sur les flux inverses seuls. C'est le cas des études faites par : Crainic *et al.* (1993) [39], Krikke *et al.* (1998) [100], Spengler *et al.* (1997) [142] et Gupta et Taleb (1994) [82]. La seconde orientation possible est de traiter la combinaison des flux inverses et traditionnels comme l'ont fait Duhaime *et al.* (2001) [56], Feillet *et al.* (2002) [59], Del Castillo et Cochran (1996) [43], Lu (2003) [106] et Thierry (1997) [148].

Le nombre de périodes étudiées :

Un critère qui ressort des différentes études est le nombre de périodes sur lequel est réalisé l'étude. Pour ce critère là nous avons deux possibilités, soit l'étude est mono-périodique comme dans Duhaime *et al.* (2001) [56], Lu (2003) [106], Krikke *et al.* (1998) [100] et Spengler *et al.* (1997) [142], soit l'étude est multi-périodique comme dans Crainic *et al.* (1993) [39], Del Castillo et Cochran (1996) [43], Feillet *et al.* (2002) [59], Gupta et Taleb (1994) [82] et Thierry (1997)[148].

Méthodes de résolution :

En ce qui concerne les méthodes utilisées pour résoudre les problèmes du niveau opérationnel de la logistique inverse en voici les principales :

- la programmation linéaire : Del Castillo et Cochran (1996) [43] par exemple utilisent deux programmes linéaires (un agrégé et l'autre non) et combinent ces résultats avec ceux obtenus par simulation ;
- la simulation : Del Castillo et Cochran (1996) [43] utilisent la simulation pour imiter les règles utilisées par les planificateurs, Thierry (1997) [148] utilise des modèles de simulation pour comparer les stocks de sécurité nécessaires pour garantir un niveau de service ;
- la relaxation lagrangienne : Lu (2003) [106] l'utilise pour résoudre un problème intégrant production et distribution avec gestion des flux direct et inverse simultanément ;
- la théorie des graphes : Penev et De Ron (1996) [122] utilisent un algorithme de plus court chemin afin de définir les séquences optimales de désassemblage pour récupérer certains composants des produits dans un processus de *"cannibalisation"* ;
- les heuristiques (type construction et améliorations) : Taleb et Gupta (1997) [144] proposent un algorithme en deux phases. La première pour déterminer le nombre de produits à désassembler et la deuxième pour planifier les opérations de désassemblage de ces produits.

2.2 Les tournées de véhicules

Notre étude est une application des problèmes de construction de tournées. Ce problème est très étudié dans la littérature. Nous pouvons notamment citer les états de l'art de Toth et Vigo (2002) [153] et de Cordeau *et al.* (2007) [35]. Cependant, nous trouvons souvent différentes variantes correspondant à plusieurs contraintes. Nous ne citerons dans ce rapport que les contraintes qui ont un intérêt dans notre problème. Ainsi nous verrons le

problème de construction de tournées traité avec la contrainte de fenêtre de temps, le problème de construction de tournées traité avec gestion des stocks et le problème de construction de tournées avec collectes et livraisons.

2.2.1 Le problème de tournées de véhicules avec fenêtres de visite

Comme il a été expliqué dans le chapitre précédent (1.2.2), dans certains cas une contrainte de respect de fenêtres de visite est ajoutée au problème de construction de tournées. L'un des premiers travaux sur ce sujet est celui réalisé par Pullen et Webb (1967) [125] sur un problème de construction de tournées pour le courrier de Londres.

Nous allons classer les différentes études selon les méthodes de résolution : approchées ou exactes. Dans les différentes études réalisées sur un *VRPTW*[6] résolu grâce à une méthode approchée citons tout d'abord Taillard *et al.* (1997) [143] qui utilise une méthode taboue pour résoudre un problème avec des fenêtres de temps larges (*cf.* 1.2.2). Nous pouvons également citer les travaux de Cordeau et Laporte (2001) [37] qui utilisent une méthode taboue également. De même citons les travaux de Berger et Barkaoui (2004) [15] qui utilisent des algorithmes génétiques pour résoudre ce problème. Kontoravdis et Bard (1995) [99] ont quant à eux utilisé une métaheuristique nommée *GRASP* (*Greedy Randomized Adaptive Search Procedure*). Cette méthode permet de construire une solution réalisable en faisant intervenir une part d'aléatoire, puis un processus de recherche locale est appliqué sur cette solution. Cette méthode est un processus itératif dans lequel la phase de construction de solution et la phase de recherche de voisinage sont répétées un certain nombre de fois (paramètre de la méthode). En ce qui concerne les méthodes exactes, nous pouvons citer : Desrochers *et al.* (1992) [48] qui utilisent la technique de génération de colonnes pour résoudre des instances contenant 100 clients. Dans de nombreux travaux, des bornes pour le problème de *VRPTW* ont été calculées notamment par décomposition Lagrangienne comme dans les travaux réalisés par : Madsen (1990) [108] ou Halse (1992) [83]. Ces bornes ont par la suite été utilisées dans un processus de Branch and Bound [102] afin de résoudre des instances à 100 clients.

Pour plus de précisions concernant ce problème, un état de l'art sur le problème de construction de tournées avec fenêtres de visite nous invitons le lecteur à se référer aux travaux faits par Cordeau *et al.* [34] (2002) et par Bräysy et Gendreau (2005) [21], [22].

2.2.2 Le problème de tournées de véhicules avec gestion des stocks

Lorsque des contraintes de gestion de stocks sur plusieurs jours sont ajoutées au problème du VRPTW, nous arrivons à un problème d'*IRP* (*Inventory Routing Problem*). En effet dans ce cas là, la décision à prendre concernant la définition des tournées est couplée avec la gestion des stocks des clients. Le problème de tournées avec gestion des stocks consiste à déterminer pour chaque jour de la période de planification, les tournées à effectuer par chacun des véhicules, les quantités de produits à livrer afin de minimiser la somme des coûts de transports, des coûts de stockage et des coûts fixes des véhicules utilisés tout en veillant à ne mettre aucun client en rupture.

Un autre problème connu sous le nom de *Period Vehicle Routing Problem (PVRP)* s'apparente à l'*IRP*. Dans le *PVRP* chaque client doit être visité un certain nombre de fois au cours de l'horizon de planification. Le nombre de visites de chaque client est une donnée du problème. Le but est ici de minimiser les coûts de transport en satisfaisant le nombre de visites obligatoires de chaque client.

Parmi les différentes applications concrètes du problème de construction de tournées avec gestion de stocks nous pouvons citer :

- le ramassage de containers de recyclage de papiers (Baptista *et al.*, 2002 [5]) ;
- le ramassage des déchets recyclables sur un campus (Bommisetty *et al.*, 1998 [19]) ;
- la distribution de produits pétroliers (Campbell et Savelsbergh, 2002 [24] Malépart *et al.*, 2003 [110]) ;
- la distribution de gaz industriels (Witucki *et al.*, 1997 [164]).

Le problème de construction de tournées de véhicules avec gestion de stocks est un cas particulier de problème de tournées de véhicules. C'est pourquoi, nous allons ici appliquer la classification des problèmes de tournées de véhicules proposée par Haouari (1991). Celle-ci repose sur trois critères différents qui sont : les caractéristiques de la flotte de véhicules, le type de la demande des clients et le critère d'optimisation.

[6] Vehicle Routing Problem with Time Windows, *cf.* chapitre 1

Commençons par le critère concernant la flotte de véhicules. Celle-ci est constituée de plusieurs véhicules dans tous les cas sauf pour l'étude menée par Reiman *et al.* (1999) [126] où un seul véhicule compose la flotte. Dans la littérature nous pouvons ensuite trouver deux types de flottes différentes : une flotte homogène (tous les véhicules sont de capacité identique) dans Anily (1994) [1], Chan *et al.* (1998) [27], Kim et Kim (2002) [98] ou une flotte hétérogène dans Tan et Beasley (1984) [145]. Dans la plupart des cas les véhicules doivent visiter plusieurs clients par tournée mais Burns *et al.* (1985) [23] et Taqa Allah *et al.* (2000) [146] quant à eux comparent les deux stratégies de construction de tournées : soit un seul client est visité par tournée soit plusieurs. D'une façon générale les véhicules partent chargés du dépôt, visitent tous leurs clients et reviennent à vide au dépôt. Cependant, Jaillet *et al.* (1986) [91] introduisent la notion de dépôts intermédiaires ("satellites") dans lesquels les véhicules peuvent venir se réapprovisionner en cours de tournée.

Voyons le critère suivant qui est celui de la nature de la demande des clients. Deux approches se distinguent : soit une approche déterministe où la quantité de produits à livrer pour chaque client est connue, soit une approche stochastique où la consommation des clients n'est connue qu'avec une certaine probabilité (le niveau réel du stock n'est connu qu'à l'arrivée du véhicule chez le client). Dans la littérature même si généralement c'est l'approche déterministe qui est le plus souvent adoptée comme le font par exemple Anily et Federguen (1993) [2], nous pouvons cependant souligner les différentes approches stochastiques réalisées : Federgruen et Zipkin (1984) [58], Dror et Ball (1987) [54], Trudeau et Dror (1992) [154]).

Regardons enfin le troisième critère de cette classification : les critères d'optimisation. Nous pouvons en rencontrer trois types différents :

- minimisation du coût de transport total (c'est le cas le plus répandu dans la littérature) ;
- minimisation du coût de transport et du coût de stockage : Bertazzi *et al.* (1999) [16], Kim et Kim (2002) [98] ;
- minimisation du nombre de véhicules utilisés (*c.-à-d.* de la taille de la flotte nécessaire) : Gaudioso et Paletta (1992) [69].

Dans la majorité des cas les problèmes d'IRP sont résolus avec des méthodes approchées. Un ensemble d'autres références sur le problème d'IRP va être donné plus loin dans ce chapitre (2.2.4).

Un état de l'art concernant les problèmes de constructions de tournées avec gestion des stocks a été réalisé par Campbell *et al.* (2002) [24].

2.2.3 Le problème de tournées de véhicules avec collectes et livraisons

Dans notre problème tel qu'il est décrit dans sa forme générale dans le chapitre 1 nous pouvons trouver un problème classique de la littérature qui est celui de Collectes et Livraisons. Dans ce type de problème nous trouvons deux catégories : un problème général de Collectes et Livraisons ou un *VRPPD* (*Vehicle Routing Problem with Pickups and Deliveries*). Dans la littérature plusieurs définitions sont possibles pour les deux catégories de problème de Collectes et Livraisons que nous allons définir par la suite. Nous avons décidé de choisir les deux définitions suivantes mais ce ne sont pas forcément des définitions retenues par tous.

Nous invitons le lecteur à lire les états de l'art proposés par Desaulniers *et al.* [45] (2002) et par Parragh *et al.* (2008) [120], [121] pour de plus amples renseignements sur le problème de construction de tournées avec collectes et livraisons.

Problème général de Collectes et Livraisons

Le problème général de Collectes et Livraisons (appelé aussi *"General Pickup and Delivery Problem : PDP"*) est une extension des problèmes de *VRP*. Ce problème est un problème d'optimisation où le but est de trouver un ensemble de tournées de façon à satisfaire l'ensemble des demandes de transport. Chaque demande de transport est caractérisée par une charge, un ou plusieurs points de départ (collectes) et un ou plusieurs points d'arrivée (livraisons). Pour satisfaire ces demandes de transport une flotte de véhicules est disponible. Lorsqu'une demande de transport est prise en charge par un véhicule il doit l'effectuer jusqu'au bout sans effectuer de livraison intermédiaire de cette charge sur un autre site autre que la destination de celle-ci. De même, les demandes de transport se caractérisent toujours par une demande de collecte vers un point de livraison. Deux contraintes sont donc mises en œuvre dans ce genre de problème *"Precedence"* qui oblige pour chaque demande de transport de visiter son nœud de départ avant celui d'arrivée, et *"Pairing"* qui oblige à visiter dans la même tournée le nœud de départ et d'arrivée d'une même demande de transport. Si les points de Collecte et/ou de Livraisons ont

des plages horaires de visite le problème devient un problème de collectes et livraisons avec fenêtres de visite (*PDPTW*). Si la flotte de véhicules est composée d'un véhicule nous serons dans un problème noté : *1-PDP* ou *1-PDPTW* et si plusieurs véhicules sont disponibles nous serons alors dans un problème noté : *m-PDP* ou un *m-PDPTW*.

Plusieurs cas d'applications peuvent être modélisés comme un problème de Collectes et Livraisons :

- le transport à la demande (*DARP : Dial A Ride Problem*) [38], il s'agit ici de transporter des personnes en groupe ou seul d'un point de départ à une destination en un temps maximal de transport ou entre deux instants définis. Deux objectifs peuvent être mis en conflit la minimisation des coûts liés au transport et la maximisation de la satisfaction du client en respectant les instants définis ou en ne dépassant pas le temps de trajet maximal défini ;
- le transport de personnes handicapées (Handicapped person transportation problems (HTP)) [152], il s'agit ici d'un cas particulier du *DARP* où les personnes transportées sont handicapées, ainsi des contraintes liées aux différents besoins sont ajoutées (besoin d'un véhicule particulier, besoin d'espace supplémentaire...) ;
- les problèmes de messageries (Courier Company Pickup and Delivery Problems (CCPDP) [138] consistent à distribuer des lettres et colis de leur expéditeur à leur destinataire. Chaque demande de transport est caractérisée par un site de collecte et un site de livraison plus une fenêtre horaire pour réaliser le service.

De nombreux état de l'art ont été réalisés sur le problème général de Collectes et Livraisons notons notamment : Mitrovic-Minic (1998) [114], Savelsbergh et Sol (1995) [136] ou encore Cordeau *et al.* (2007) [35]. D'autres références sur ce problème sont données dans la partie 2.2.4.

VRPPD

Plusieurs définitions sont fournies dans la littérature pour ce type de problème. En effet, pour certains le *VRPPD* a la même définition que le problème général de Collectes et Livraisons et pour d'autres le *VRPPD* correspond aux problèmes de Collectes et Livraisons mais où pour toutes demandes de transport soit l'origine soit l'arrivée est représentée par le dépôt. Nous avons décidé d'opter pour la seconde définition comme le font Nagy et Salhi (2005) [117].

Ce problème est un problème d'optimisation dont le but est de trouver un ensemble de tournées de façon à satisfaire l'ensemble des demandes de transport. Chaque demande de transport est caractérisée par une charge, un ou plusieurs points de départ (collectes) et un ou plusieurs points d'arrivée (livraisons). Aucune collecte et livraison ne peuvent se faire entre sites au sein du réseau. Lorsque nous rajoutons des fenêtres de visite à un problème de *VRPPD* nous obtenons un *VRPPDTW* : *Vehicle Routing with Pickups and Deliveries and Time Windows*. Nous distinguons trois modèles différents pour le *VRPPD* :

- Tout d'abord, nous avons *"Delivery-first, pickup-second VRPPD"*. Dans ce type de modèle les sites sont classés en deux catégories : les sites qui reçoivent des produits et les sites qui envoient des produits. Chaque véhicule ne pourra collecter des produits qu'après avoir fini toutes ses livraisons ;
- Ensuite nous avons *"Mixed pickups and deliveries"*. Dans ce cas là, les collectes et les livraisons sont réalisées sans ordre précis ;
- Finalement, le troisième modèle est *"Simultaneous pickups and deliveries"*. Dans ce cadre là les sites peuvent simultanément recevoir et envoyer des produits.

Notre problème correspond plus ici à un *VRPPDTW* qu'à un problème général de Collectes et Livraisons. Nous pouvons de plus préciser que notre problème s'apparente plus à un problème de *VRPPDTW* avec collecte et livraison simultanées.

2.2.4 Le problème de tournées de véhicules étudié

Nous n'avons trouvé dans la littérature qu'une étude pouvant se rapprocher de nos travaux. Il s'agit de celle décrite dans les travaux de Christiansen et Nygreen (1998) [31] et Christiansen (1999) [30]. Ceux-ci seront décrits dans le paragraphe évoquant les méthodes de résolution exactes pour résoudre notre problème.

Dans la suite de cette partie nous allons donner quelques méthodes de résolution qui ont été employées sur des problèmes en lien avec notre problème. Comme il a été dit précédemment seul le paragraphe sur les méthodes exactes évoque une étude similaire à la nôtre.

Les méthodes approchées

Des travaux sur la résolution d'*IRP* ont pu être faits mais dans tous les cas que nous avons trouvés il ne s'agit que des demandes de livraisons ou que des demandes de collectes avec gestion des stocks, les deux flux ne sont pas traités simultanément. Plusieurs travaux sur la résolution heuristique de l'*IRP* ont été réalisés citons notamment les travaux de Taqa allah *et al.* (2000) [146] qui résolvent le problème de l'*IRP* pour plusieurs produits grâce à des heuristiques de type glouton. Nous pouvons citer également les travaux de Malépart *et al.* (2003) [110] qui développent des heuristiques constructives pour un problème où deux types de stations cohabitent : celles dont le stock est géré par les transporteurs et celles qui lancent leurs commandes lorsqu'il est nécessaire.

De même, des travaux sur la construction de tournées de manière approchée comprenant simultanément des collectes et des livraisons ont pu être faits mais sur un aspect mono-périodique et sans gestion des stocks. Nous pouvons citer les travaux de Bianchessi et Righini (2007) [17] sur des heuristiques de construction et améliorations pour le problème de construction de tournées comprenant simultanément des collectes de déchets et des livraisons de produits. Pour cela, ils introduisent les deux concepts de "séquence fortement réalisable" (séquence de clients pouvant être visités par un véhicule sans violer la capacité du véhicule) et "séquence faiblement réalisable" (séquence de clients pouvant devenir une séquence fortement réalisable par un changement d'ordre de visite). Ils utilisent plusieurs méthodes de recherche locale : avec voisinages simples, avec voisinages complexes, avec voisinages variables et utilisent également une recherche taboue reprenant les différentes recherches locales nommées précédemment. De plus, nous pouvons citer les travaux de Mosheiov (1998) [115] qui considère comme nous que chaque demande (de livraison ou de collecte) peut être honorée par plus d'un véhicule et qui résout le problème grâce à deux heuristiques basées sur le principe du "Route First, Cluster Second". Citons également, les travaux de Tan et Beasley (1984) [145] et de Dror et Ball (1987) [54] sur un problème d'*IRP* qui se basent sur les résultats de Fisher et Jaikumar (1978) [64]. Dans ces travaux la résolution du problème de voyageur de commerce est effectuée par des algorithmes classiques. Les techniques heuristiques peuvent être appliquées à partir d'une solution réalisable (Christofides et Beasley, 1984 [32]) ou elles peuvent aussi servir à obtenir une solution de départ et ensuite l'améliorer (Gaudioso et Paletta, 1992 [69]). Notons l'étude menée par Toth et Vigo (1997) [152] qui proposent des méthodes d'insertion parallèles pour le *m-PDPTW*. Une autre technique de résolution approchée est la méthode de décomposition. Citons Dumas *et al.* (1989) [57] qui introduisent la notion de mini-cluster pour résoudre un problème de grande taille. Une autre classe de méthodes de résolution est constituée des algorithmes d'amélioration qui partent d'une solution initiale et qui vont chercher à l'améliorer. Une de ces méthodes est la recherche locale. C'est celle-ci qu'ont utilisé Van-Der-Bruggen *et al.* (1993) [156] pour résoudre le *1-PDPTW*. Enfin, voyons les métaheuristiques. En effet nous pouvons citer le recuit simulé (Van-Der-Bruggen *et al.*, 1993 [156] pour un 1-PDPTW), la recherche taboue (Malca et Semet, 2004 [109] pour un m-PDPTW), les colonies de fourmis (pour un m-PDPTW) ou les algorithmes génétiques (Pankratz, 2005 [119] pour un m-PDPTW) qui ont pu être entre autre utilisés. Un problème qui se rapproche le plus de celui que l'on considère est le VRPSDP : *Vehicle Routing Problem with Simultaneous Delivery and Pick-up*. Halskau *et al.* (2001) [84], Hoff et Lokketangen (2006) [88] traitent de ce problème en adoptant une hypothèse qui nous intéresse particulièrement : la possibilité pour un même véhicule de visiter en deux fois certains sites (tournées en lasso). Halskau *et al.* (2001) [84] utilisent des méthodes heuristiques pour résoudre ce problème, alors que Hoff et Lokketangen (2006) [88] utilisent la méthode taboue.

Les méthodes exactes

Voyons tout d'abord les méthodes de résolution exactes pour les problèmes de collectes et livraisons. Notons tout d'abord les modèles de programmation linéaires en nombres entiers. Parmi les différentes études proposant une modélisation mathématique pour le problème de Collectes et Livraisons avec fenêtres de visite (*PDPTW*), notons les études de Lau et Liang (2001) [104] pour le *1-PDPTW* et de Savelsbergh (1995) [136] pour le *m-PDPTW*. Une deuxième catégorie de méthodes de résolution exacte est la programmation dynamique, comme par exemple dans les travaux de Desrosiers *et al.* (1986) [50]. Une autre méthode de résolution est la méthode de séparation et évaluation. Nous pouvons ici citer à titre d'exemple les travaux de Kalantari *et al.* (1985) [93] pour résoudre le *1-PDPTW*. Des techniques de génération de colonnes basées sur des modèles de recouvrement ou de partitionnement ont également été utilisées, citons les travaux de Velasco *et al.* (2005) [160] qui résolvent le problème général de Pick-up and Delivery pour un problème de tournées d'hélicoptères. Enfin un algorithme de Branch and Cut peut également être utilisé pour résoudre ce genre de problème. Cordeau (2003) [36] résout

un problème de *m-PDPTW* avec ce type de méthode.

Les méthodes exactes ne sont pas utilisées pour résoudre le problème d'*IRP* entièrement. En effet, dans la littérature, seules des parties de ce problème sont résolues optimalement. Nous pouvons voir dans certains travaux l'emploi de la relaxation lagrangienne pour obtenir une borne inférieure de la solution (Bell *et al.*, 1983 [12]). Notons aussi que la décomposition de Benders est utilisée dans les travaux de Federgruen et Zipkin (1984) [58].

Nous n'avons trouvé dans la littérature qu'une étude pouvant se rapprocher de nos travaux. Il s'agit de celle expliquée dans Christiansen et Nygreen (1998) [31] et Christiansen (1999) [30]. Dans ces travaux les auteurs étudient un problème sur la planification de navires entre 15 ports sur une durée d'un mois. Les caractéristiques de leur problème montrent une certaine ressemblance avec le nôtre : multi-véhicules, chargement et déchargement de produits, présence de fenêtres de visite, un produit et gestion des stocks. Les auteurs utilisent une technique de génération de colonnes avec Branch and Price (Barnhart *et al.*, 1998 [7] et Vanderbeck, 2000 [159]). Les auteurs modélisent leur problème avec deux sous-problèmes : un pour la construction de tournées des navires et un autre pour la gestion des stocks. Le but de l'étude est de minimiser les coûts de routage uniquement. Les différences remarquées avec notre problème sont les suivantes : il n'y a pas de présence de dépôt c'est-à-dire que les tournées n'ont pas de point fixe de départ et d'arrivée, de plus souvent les quantités nécessaires pour chaque port impliquent la présence d'un seul port dans la tournée.

Nous invitons le lecteur à lire l'article de Baldacci *et al.* (2007) [4] sur les avancées des algorithmes exactes pour les problèmes de construction de tournées.

2.3 Les méthodes hybrides (RO/PPC) dans la logistique et le transport

Dans cette partie de notre état de l'art, nous allons présenter les études utilisant la programmation par contraintes[7], et les méthodes hybrides pour résoudre un problème de construction de tournées. Les deux approches, programmation par contraintes et programmation linéaire, permettent de résoudre un grand nombre de problèmes d'optimisation combinatoire. Utiliser la complémentarité des deux approches semble une idée intéressante à poursuivre au vu des différents résultats obtenus dans les travaux cités après. C'est dans le but tout d'abord de comparer les méthodes purement issues de la recherche opérationnelle avec les méthodes hybrides mais aussi afin de valider l'efficacité des méthodes hybrides sur le type de problème que nous traitons, que nous allons nous employer à utiliser des méthodes hybrides.

Tout d'abord voyons l'approche utilisée par Rousseau *et al.* (2002) [132]. Dans cette étude les auteurs utilisent *la programmation par contraintes pour résoudre les sous-problèmes de la génération de colonnes*. L'originalité de cette étude réside dans le fait qu'ils tentent de résoudre un problème cyclique. Ainsi, vont ils résoudre le problème du plus court chemin élémentaire par de la programmation par contraintes, pour la résolution de *VRPTW* par génération de colonnes. Leurs recherches ont abouti au développement d'une nouvelle contrainte pour la résolution de problème de routage, la contrainte *CBC*[8], qui peut être utilisée pour résoudre des problèmes de *TSP* ou de *VRP*. De plus, ils ont introduit trois nouveaux algorithmes d'élimination d'arcs, utiles pour résoudre les problèmes de plus courts chemins de coût réduit négatif dans les méthodes de génération de colonnes ou de décomposition Lagrangienne.

Dans l'étude menée par Caseau et Laburthe (1999) [26], les auteurs proposent une méthode pour la résolution de *VRP* de grande taille (des milliers de clients et des centaines de véhicules). Ainsi, ils utilisent une méthode qui garde les avantages des méthodes d'insertion (flexibilité et adaptabilité) mais qui offre des améliorations significatives en ce qui concerne les qualités de solutions. Comme leur heuristique est rapide, ils utilisent *LDS*[9] [87] pour les problèmes de taille moyenne.

> *On appelle divergence un choix de valeur dans l'arbre de recherche qui ne correspond pas à ce que l'heuristique de choix prescrit. Ainsi, LDS permet le parcours de l'arbre de recherche en autorisant un certain nombre de choix différents (divergences) du choix de l'heuristique. Le nombre de divergences maximales autorisées est un paramètre de la méthode. LDS commence par explorer les*

[7] Des rappels de base de la programmation par contraintes et la programmation linéaire sont faits respectivement en annexe 1 et 2.
[8] Can Be Connected
[9] Limited Discrepancy Search

chemins suivant les choix de l'heuristique (ayant donc 0 divergence), puis augmente peu à peu le nombre de divergences autorisées afin d'arriver au nombre maximum de divergences.

Leur heuristique est, quant à elle, basée sur la meilleure insertion possible grâce à la fonction d'évaluation suivante : le coût d'insertion de i entre a et b est $c(a,i) + c(i,b) - c(a,b)$. Puis, au lieu d'utiliser une recherche locale une fois qu'une solution est construite, ils appliquent le principe d'optimisation locale progressive. Ainsi, après chaque insertion une ré-optimisation grâce au 3-opt est effectuée. Une extension de leur heuristique a été réalisée pour le *VRPTW*. Ils concluent leur étude en notant que la technique *LDS* doit être employée avec une heuristique rapide, car elle possède un temps de calcul assez conséquent.

Voyons maintenant l'étude de Shaw (1998) [137]. Dans cet article, l'auteur utilise une méthode de recherche locale appelée *LNS*[10]. Cette technique explore un grand voisinage de la solution actuelle en choisissant un certain nombre de visites à enlever de la solution, puis ré-insère ces visites en utilisant un arbre de recherche contraint pour évaluer le coût et la validité de la modification. L'auteur utilise également la technique *LDS* (expliquée ci-dessus), mais il l'utilise dans la phase de ré-insertion grâce à son arbre de recherche. Le nombre de visites qui est enlevé est changé dynamiquement et le choix des sites qui doivent être enlevés est effectué aléatoirement. Tout ceci entraîne une certaine diversité des solutions obtenues. L'auteur utilise sa méthode de résolution sur un *VRP* et sur un *VRPTW*. Des résultats comparables à ceux obtenus par des métaheuristiques ont été obtenus.

Dans l'article écrit par De Backer *et al.* (2000) [42], les auteurs introduisent une méthode qui utilise des techniques d'amélioration itérative et des metaheuristiques avec des structures de programmation par contraintes (ILOG Solver). Ils appliquent cette méthode à des problèmes de construction de tournées. Ainsi, afin d'éviter les pièges des minima locaux, la technique d'amélioration itérative sera couplée à une métaheuristique. Deux métaheuristiques ont été testées dans cet article : la recherche taboue (Glover (1989 [73], 1990 [74]) et la méthode nommée *Guided Local Search* (Voudouris *et al.* 1995 [163], Voudouris 1997 [162]). La méthode *Guided Local Search* ajoute des pénalités à la fonction objectif en fonction des solutions visitées précédemment. La recherche est pénalisée si les solutions sont trop proches des minima locaux rencontrés. La contrainte de chemin du solver ILOG [89] a été utilisée pour propager l'espace libre dans les véhicules et l'instant de visite des clients pour chaque tournée. Cette contrainte permet de vérifier que les quantités (temps, charge *etc.*) accumulées au cours d'un chemin respectent les bornes. Ainsi si on note R_i le successeur de i dans le chemin, q_i la quantité accumulée en i et q_{ij} la quantité ajoutée au parcours de l'arc (i,j), la contrainte assure que $R_i = j \Rightarrow Q_j \geq Q_i + q_{ij}$. La recherche est effectuée par les techniques d'amélioration itérative. Lorsque cette technique a besoin de vérifier la validité d'une solution potentielle elle appelle la structure de programmation par contraintes. Ainsi, afin d'effectuer la propagation des décisions et de vérifier la faisabilité de la solution potentielle, l'ensemble des variables de décision R est instancié, ainsi les variables de temps et de capacité ont leur domaine qui se réduisent. Si une variable se retrouve avec son domaine vidé alors la solution n'est pas réalisable. Quatre améliorations itératives ont été étudiées : 2-opt, le déplacement d'une visite au sein d'une même tournée ou d'une autre tournée, un échange de deux visites au sein d'une même tournée ou entre deux tournées ou l'échange de la fin de deux tournées.

Dans leurs travaux, Domenjoud *et al.* (1998) [52] proposent une solution pour un problème de construction de tournées avec collecte et dépôt, et établissent le planning de travail des conducteurs en fonction. Ce problème correspond à un problème de transport de personnes handicapées pour l'entreprise : GIHP[11]. Les contraintes qui sont prises en compte dans leur problème de construction de tournées sont la capacité du véhicule et les fenêtres de temps. Voici en quelques chiffres les données du problème : 20 véhicules, 20 conducteurs et une moyenne de 200 à 300 demandes de transport par jour. Les demandes de transport sont faites par avance et spécifient le lieu de départ, le lieu d'arrivée, les contraintes horaires. Le but est de minimiser le coût total. Le projet a été divisé en trois parties : la construction des tournées, l'affectation des conducteurs aux tournées et l'affectation des véhicules aux tournées. Pour la construction de tournées une contrainte de permutation et une contrainte de temps sont utilisées pour trouver l'ordre total de visite des sites. Ensuite, une contrainte de partitionnement, une contrainte de disponibilité, des contraintes liées à la législation du travail et une contrainte de temps sont utilisées pour affecter un conducteur et un véhicule à une tournée. Finalement, pour l'affectation des véhicules aux tournées ils utilisent une contrainte de partitionnement et une contrainte de capacité. Il y a ensuite deux modes d'optimisation. La première méthode est une optimisation globale (Branch and Bound, [102]). La seconde méthode consiste à trouver une solution avec l'optimisation globale pour un sous-ensemble de demandes, puis

[10] Large Neighbourhood Search

[11] Groupement pour l'insertion des personnes handicapées physiques.

appliquer un algorithme incrémental qui itérativement propage les contraintes pour insérer une demande non planifiée parmi celles déjà planifiées.

2.4 Conclusions

Dans ce chapitre nous avons passé en revue les différents problèmes qui touchent de près ou de loin à notre problème. Ainsi nous avons constaté que la logistique inverse est un sujet très étudié. L'ensemble des trois niveaux de la planification hiérarchisée a été traité. Toutefois, l'aspect construction de tournées et gestion des stocks dans un contexte de logistique inverse n'a pas été vraiment traité. De plus au vu des différents résultats qui ont pu être obtenus sur des problèmes d'*IRP*, il était intéressant d'approfondir l'idée de prise en compte des stocks. Les méthodes hybrides alliant techniques de recherche opérationnelle et de programmation par contraintes montrent dans les études que nous avons pu citer des résultats encourageants et confirment la complémentarité de ces deux techniques. C'est pourquoi nous avons décidé d'utiliser de telles méthodes dans la suite de nos travaux afin de les comparer à des techniques purement empreintes à la recherche opérationnelle et afin de voir si elles peuvent être propices à la résolution de notre problème.

Nous allons dans la suite résoudre notre problème grâce à des méthodes classiques de construction et améliorations sur lesquelles nous allons tester diverses techniques et combinaisons de techniques afin d'en identifier la meilleure. Nous allons résoudre ensuite notre problème grâce à une métaheuristique nommée GRASP (Greedy Randomized Adaptive Search Procedure), qui utilisera des techniques d'amélioration identifiées auparavant et des techniques issues de la programmation par contraintes. Finalement, nous allons résoudre notre problème de façon exacte et ce, grâce à la technique de génération de colonnes. Dans cette méthode nous testerons diverses façons de résoudre le sous-problème : programmation dynamique, recherche taboue combinée à la programmation dynamique et techniques de programmation par contraintes.

Chapitre 3

Jeux de Données

Dans ce chapitre nous allons décrire les différentes instances que nous avons utilisées dans nos travaux. Nous commencerons par voir les différentes instances connues de la littérature sur différents problèmes de construction de tournées de véhicules. Ensuite, nous décrirons dans un premier temps les instances que nous avons générées aléatoirement avec comme base des instances connues, puis, celles provenant de cas réels et fournies par la société alfaplan Management Software & Consulting GmbH.

3.1 Dans la littérature

Dans la littérature, des fichiers d'instances pour divers problèmes de construction de tournées existent, en voici une liste non exhaustive :

- Les constructions de tournées sous contrainte de capacité *(CVRP)* : Augerat *et al.*, Van Breedam, Christofides et Eilon, Rinaldi et Yarrow, Taillard ...
- Les constructions de tournées sous contrainte de capacité et avec fenêtre de visite *(CVRPTW)* : Van Breedam, Cordeau, Solomon ...
- Les constructions de tournées avec collectes et livraisons *(VRPPD)* : Van Breedam ;
- Les constructions de tournées avec collectes et livraisons et fenêtre de visite *(CVRPPDTW)* : Van Breedam, Reinelt ;
- Les constructions de tournées avec plusieurs dépôts *(MDVRP)* : Cordeau, Gillet et Johnson ;
- Les constructions de tournées avec plusieurs dépôts et des fenêtres de visite *(MDVRPTW)* : Cordeau ;
- Les constructions de tournées périodiques *(PVRP)* : Cordeau ;
- Les constructions de tournées périodiques avec fenêtres de visite *(PVRPTW)* : Cordeau ;
- Les constructions de tournées avec demandes préemptives *(SDVRP)* : Cordeau ;
- Les constructions de tournées avec demandes préemptives et fenêtres de visite *(SDVRPTW)* : Cordeau.

Toutes ces instances sont disponibles pour la plupart sur Internet ainsi que les meilleurs résultats obtenus (notamment sur : http ://neo.lcc.uma.es/radi-aeb/WebVRP/).

3.2 Les instances utilisées

Pour nos travaux nous avons utilisé des données existantes pour un problème proche du nôtre comme base d'instances et nous avons généré les données manquantes. Nous utilisons comme base les instances de Solomon [139] pour le problème de CVRPTW (Capacitated Vehicle Routing Problem with Time Windows) avec 25 clients. Nous conservons les données concernant les sites : fenêtres de visite, coordonnées, temps de service. Nous générons ensuite, pour chaque site les valeurs de ses stocks initiaux (de palettes vides, de produits neufs et de produits en retour), les valeurs de ses demandes en produits neufs sur les jours de planification et les valeurs des nombres de retours de produits pour chaque journée. De façon à coller à la réalité nous avons décidé de créer 3 types de site : les petits, les grands et les très grands. En effet, nous nous sommes placés dans la situation d'une enseigne de la grande distribution qui décline ses offres auprès du public à travers différentes tailles de magasins.

Nous travaillons sur un horizon de 5 jours correspondant à une semaine de travail. Sur chaque palette, nous disposons de 4 produits. Nous considérons que les 4 produits sont livrés sur une palette indivisible. Ce chiffre a été choisi dans le but d'être le plus proche de la réalité. En effet, nous nous sommes basés sur la réglementation sur les DEEE (*cf.* chapitre 1), 4 produits sur une palette est le nombre moyen qui nous semblait le plus plausible. Ainsi toutes les ventes de 4 produits une palette vide est disponible pour être collectée.

3.2.1 Les caractéristiques des trois types de sites

Les chiffres proposés ici pour les différentes catégories de sites sont le fruit d'une estimation de ce que peut être la réalité pour une enseigne de grande distribution.

- Les petits sites :
  - capacité de stockage de 50 ;
  - une demande journalière fixe de 20 et une demande suivant une loi normale de moyenne 20 et d'écart type 5 et une demande suivant une loi normale de moyenne 20 et d'écart type 15 ;
  - un taux de retour de 2 par jour ;
  - stock de palettes vides est de 5 au début de la simulation ;
  - stock de produits en retour est de 2 au début de la simulation ;
- Les grands sites :
  - capacité de stockage de 100 ;
  - une demande journalière fixe de 40 et une demande suivant une loi normale de moyenne 40 et d'écart type 5 et une demande suivant une loi normale de moyenne 40 et d'écart type 15 ;
  - un taux de retour de 4 par jour ;
  - stock de palettes vides est de 10 au début de la simulation ;
  - stock de produits en retour est de 4 au début de la simulation ;
- Les très grands sites :
  - capacité de stockage de 150 ;
  - une demande journalière fixe de 80 et une demande suivant une loi normale de moyenne 80 et d'écart type 5 et une demande suivant une loi normale de moyenne 80 et d'écart type 15 ;
  - un taux de retour de 8 par jour ;
  - stock de palettes vides est de 20 au début de la simulation ;
  - le stock de produits en retour est de 8 au début de la simulation.

3.2.2 Les catégories des instances

Les instances de Solomon utilisées sont constituées de 56 fichiers. Ils sont classés en 6 catégories : R1, R2, C1, C2, RC1 et RC2. Chaque catégorie contient entre 8 et 12 instances. Dans les catégories R1 et R2 les données géographiques sont générées aléatoirement selon une distribution uniforme. Dans les catégories C1 et C2 les clients sont placés en "cluster". Et dans les catégories RC1 et RC2 quelques clients sont placés en "cluster" et d'autres aléatoirement. Dans les fichiers des catégories R1, C1 et RC1 l'horizon de temps est court permettant des tournées de 5 à 10 clients, tandis que dans les fichiers des catégories R2, C2 et RC2 les problèmes ont des horizons de temps plus grands qui permettent de visiter une trentaine de clients.

Pour ces 56 fichiers à 25 sites proposés par Solomon nous allons donc générer des instances exploitables pour notre problème qui auront certains profils. Nous allons générer 11 profils différents, pour chacun des profils nous aurons 56 fichiers ayant une demande fixe, 56 fichiers ayant une demande suivant une loi de GAUSS d'écart type 5 et 56 fichiers ayant une demande suivant une loi de GAUSS d'écart type 15, soit 1848 fichiers au total. Les profils que nous avons voulu étudier sont les suivants donnés dans le tableau 3.1.

Profil	% petit	% grand	% très grand
A	100	0	0
B	0	100	0
C	0	0	100
D	34	33	33
E	5	25	70
F	25	25	50
G	50	25	25
H	70	25	5
I	25	50	25
J	25	70	5
K	5	70	25

TAB. 3.1 – Catégories d'instances du problème

Les trois premiers profils (A, B et C) sont constitués d'un type unique de sites. Le profil D a une répartition équitable de tous les types de sites. E et F sont majoritairement constitués de très grands sites. G et H sont quant à eux majoritairement faits de petits sites. Et enfin, I, J et K sont constitués en majorité par des moyens sites. Dans les profils, E, F, G, H, I, J et K les deux types de sites non majoritaires sont répartis de façon à occuper la deuxième et la troisième place dans l'ordre des répartitions et vice versa.

Dans la suite de ce rapport, nous ferons référence à ces différentes catégories d'instances sous les dénominations : AFixe-25 pour le profil A ayant des demandes fixes chaque jour, AGauss5-25 pour le profil A ayant des demandes suivant une loi normale d'écart type 5, AGauss15-25 pour le profil A ayant des demandes suivant une loi normale d'écart type 15 *etc.*

3.2.3 Les instances réduites

Pour certaines méthodes de résolution (principalement les méthodes de résolution exacte), nous avons eu besoin d'instances plus petites que celles ayant 25 sites. C'est pourquoi, nous avons créé des instances avec 5 et 6 sites. Pour cela, nous avons sélectionné pour chaque instance de Solomon 5 (ou 6) sites et sur la même base que ce qui a été présenté ci-dessus nous avons généré les valeurs manquantes sur le même schéma de profil. Pour les instances à 5 sites nous utilisons les profils déjà expliqués ci-dessus (3.1) suivantes : A, B, C, E, H et J et nous avons créé les catégories L (40 % de petits et grands sites et 20 % de très grands sites), M (40 % de petits et très grands sites et 20 % de grands sites) et N (20 % de petits sites et 40 % grands sites et très grands sites). Pour les instances à 6 sites nous avons utilisé les catégories : A, B, C, D, E, H, J et K.

3.3 Les instances issues des cas réels

La société alfaplan Management Software & Consulting GmbH éditeur allemand de logiciel de construction de tournées, nous a fourni deux instances de tests issues de cas réels. Ces deux instances seront traitées grâce à nos méthodes heuristiques.

La première instance utilise une flotte de véhicules homogènes, 300 sites de livraison émettent des demandes de livraison sur 5 périodes et chaque site possède une fenêtre de visite large. Les véhicules ont une capacité limitée. Nous noterons cette instance *IR1*.

La deuxième instance utilise une flotte de véhicules homogènes, elle est composée de 216 sites de collectes, les demandes de collecte sont à effectuer sur une journée et chaque site possède une fenêtre de visite large. Les véhicules ont une capacité limitée. Nous noterons cette instance *IR2*.

Deuxième partie

Méthodes d'optimisation des problèmes de tournées de véhicules dans un contexte de logistique inverse

Chapitre 4

Résolution par des méthodes de construction et améliorations

Dans ce chapitre nous allons expliquer comment nous résolvons, grâce à des heuristiques de construction et améliorations, notre problème de construction de tournées dans un contexte de logistique inverse. Les tournées que nous construisons combinent des flux directs (du magasin vers les clients) et des flux indirects (des clients vers le magasin) tout en optimisant la gestion des stocks des clients sur plusieurs jours. Nous testons deux stratégies : une visant à répondre à la demande le jour où elle doit être honorée, c'est à dire en *juste à temps* (le magasin commande par lui-même son approvisionnement et l'approvisionnement doit se faire le jour où le client en a besoin, il n'y a pas de possibilité de livrer par avance les commandes même si cela est plus avantageux) ; l'autre autorisant à prendre de l'avance sur les livraisons, donc avec *gestion des stocks*, (on doit dans ce cas là payer un coût de stockage) si cela diminue le coût global.

Afin d'identifier la meilleure méthode de construction, nous allons éprouver deux méthodes de construction, qui sont les deux méthodes les plus utilisées et ayant donné les meilleurs résultats dans la littérature : *"meilleure insertion"* et *"plus mauvaise insertion"*. Nous testons ensuite, plusieurs méthodes d'améliorations et plusieurs combinaisons de ces méthodes. Nous concluons enfin en identifiant une méthode efficace de construction et améliorations pour chacune des deux stratégies envisagées pour notre problème.

4.1 Les méthodes de construction

Selon Laporte et Semet [103], il existe deux principales techniques pour construire une solution pour un problème de construction de tournées : la fusion de tournées déjà existantes avec un critère de gain et l'affectation au fur à mesure des nœuds aux tournées en utilisant un coût d'insertion. N'ayant pas de solutions initiales pour pouvoir effectuer les fusions, nous allons pour notre part utiliser la méthode d'affectation des nœuds.

Nous avons utilisé deux méthodes de construction de solutions, qui sont des méthodes gloutonnes classiques de la littérature : Meilleure Insertion [139] et Plus Mauvaise Insertion. "Meilleure Insertion" est une méthode de construction très connue. Quant à "Plus Mauvaise Insertion" nous l'utilisons comme méthode de construction de solution car nous avons dans l'optique de poursuivre nos travaux en résolvant ce problème à l'aide de la métaheuristique GRASP (Greedy Randomized Adaptive Search Procedure) qui est utilisée dans l'article de Kontoravdis et Bard [99] avec cette méthode de construction justement.

Les deux méthodes de construction utilisées sont des méthodes heuristiques dynamiques puisque les coûts sont révisés en fonction du problème résiduel après chaque insertion.

4.1.1 Meilleure Insertion (MI)

Nous avons choisi d'utiliser comme première méthode de construction d'une solution la méthode "Meilleure Insertion" introduit par Solomon [139] puis améliorée par Potvin et Rousseau [124].

Deux méthodes de meilleure insertion sont possibles : séquentielle ou parallèle. Nous allons utiliser la méthode parallèle, en effet les tournées vont être construites toutes en même temps et non pas une d'abord puis ensuite les autres. Nous préférons cette méthode puisque notre but n'est pas de minimiser la flotte utilisée mais

de minimiser le coût. Ainsi, si l'insertion la moins coûteuse implique la création d'une tournée alors cette tournée est créée.

Nous débutons donc notre méthode de construction par autant de tournées vides (dépôt→ dépôt) que de véhicules disponibles pour chaque journée de la planification. Pour une instance avec 25 véhicules sur une période de planification de 5 jours, nous avons donc au début 125 tournées vides.

Pour chaque demande nous cherchons quelle est la tournée dans laquelle il faut l'insérer, et au sein de cette tournée nous cherchons à quel endroit nous devons l'insérer (*i.e.* entre quels sites déjà présents dans la tournée) et quelle part de la demande (quantité) est prise en charge par cette tournée, de façon à ce que l'on ait le meilleur coût. Une fois que pour chaque demande non encore planifiée nous avons calculé le meilleur triplet : tournée, placement dans la tournée et quantité qui offre le coût minimum d'insertion de cette demande, nous choisissons d'insérer la demande ayant le plus faible "meilleur coût d'insertion" à l'endroit et pour la quantité définie par son triplet. Nous réitérons ce processus tant qu'il reste des demandes non planifiées.

Le choix de la quantité desservie par la tournée pour la demande insérée constitue une adaptation de la méthode *"best insertion"* qui normalement consiste à insérer chaque demande (valeur fixe et connue) dans chaque tournée afin de déterminer l'insertion minimum.

L'insertion d'un site dans une tournée est valide si elle respecte : la fenêtre de visite du site inséré et celles des sites déjà présents dans la tournée, les journées de réalisation de la demande (à chaque demande correspond un jour, elle doit donc être affectée à une tournée qui est réalisée ce jour là) et la capacité du véhicule. Le coût d'insertion d'un site dans une tournée comprend : le coût de création de la tournée s'il s'agit du premier site qui est inséré dans cette tournée et le coût de routage de ce site dans cette tournée (*i.e.* la distance séparant ce site de son successeur et de son prédécesseur multiplié par le coût de transport).

4.1.2 Plus Mauvaise Insertion (PMI)

Comme deuxième méthode de construction de solutions nous avons choisi de prendre la technique de la "Plus Mauvaise Insertion" afin d'insérer en priorité les demandes qui constituent le plus fort coût d'insertion. Le mécanisme de construction est le même que celui décrit dans le paragraphe 4.1.1, sauf pour la sélection de l'élément à insérer dans la solution courante, nous choisissons l'élément engendrant le plus fort surcoût. Tout comme pour la première méthode, l'insertion d'un élément dans la solution répond aux mêmes critères de validité.

4.2 Les méthodes d'amélioration

Après avoir trouvé une solution réalisable (on estime qu'il existe une solution réalisable pour toutes les instances traitées) à notre problème avec les méthodes de construction, nous explorons le voisinage de cette solution grâce aux méthodes d'amélioration.

Ce type de méthode commence à partir d'une solution réalisable, qui est successivement remplacée par une solution meilleure appartenant à son voisinage ; un voisinage N associe à chaque solution S un sous-ensemble $N(S)$ de solutions. La solution S est un optimum local par rapport au voisinage $N(S)$ s'il n'existe pas de solution strictement meilleure que S dans $N(S)$.

Nous avons défini pour notre problème six types de voisinage qui correspondent donc à six heuristiques d'amélioration. Dans tous les cas nos méthodes d'amélioration doivent vérifier avant de valider un changement que les fenêtres de visites des sites et des sites des tournées mises en cause sont respectées, que la capacité des véhicules effectuant les tournées traitées est en tout point respectée et enfin que le changement apporte un gain au niveau du coût de la solution.

Afin de décrire nos méthodes d'améliorations, nous utilisons ici la classification des voisinages proposée par Laporte et Semet [103], qui distinguent les améliorations mono-tournée et celles multi-tournées.

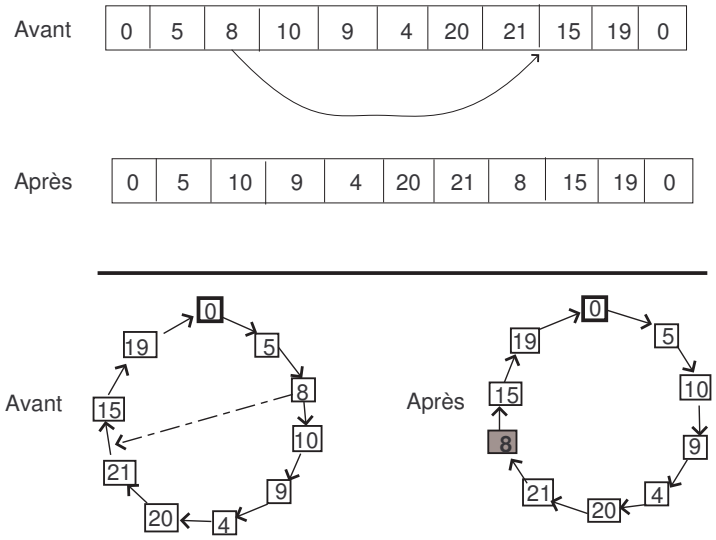
4.2.1 Les méthodes d'amélioration mono-tournée

Dans cette partie, nous allons décrire les différentes heuristiques d'amélioration mono-tournée (*i.e.* au sein d'une même tournée) utilisées.

Or-Opt

Nous utilisons au sein de chaque tournée la méthode Or-opt [118] qui consiste à déplacer une séquence de 3, 2 ou 1 nœuds consécutifs au sein d'une tournée. Pour notre étude, nous avons choisi de déplacer des séquences de un nœud. Nous choisissons de ne pas déplacer plus de nœuds car nos sites ont des fenêtres de visite donc en déplaçant plus de nœuds nous risquerions de nous trouver très fréquemment avec des solutions non réalisables. Cette méthode sera nommée par la suite : D_i (pour Déplacement dans une tournée). La figure 4.1 montre le fonctionnement de cette méthode.

Figure 4.1 *Méthode d'amélioration : D_i*



2-échange

D'autre part nous utilisons le 2-échange de nœuds au sein d'une même tournée. Cette méthode consiste à échanger deux nœuds qui sont planifiés au sein d'une même tournée. Cette méthode sera appelée : E_i (pour Échange dans une tournée). La figure 4.2 montre le fonctionnement de cette méthode.

2-Opt

Enfin comme dernière méthode d'amélioration mono-tournée, nous avons également utilisé le 2-opt, qui consiste à échanger 2 arcs d'une tournée puis à re-connecter les morceaux de cette tournée. L'abréviation utilisée pour cette méthode sera : O (pour 2-Opt). La figure 4.3 montre le fonctionnement de cette méthode.

4.2.2 Les méthodes d'amélioration multi-tournées

Nous allons ici décrire les méthodes d'amélioration multi-tournées (*i.e.* entre plusieurs tournées) proposées dans les travaux de Van Breedam [155] qui ont été ensuite reprises par Laporte et Semet [103].

Figure 4.2 *Méthode d'amélioration : E_i*

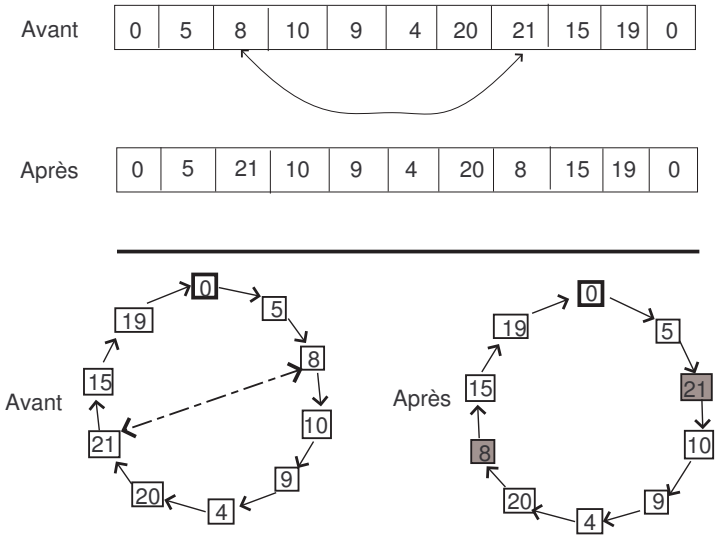
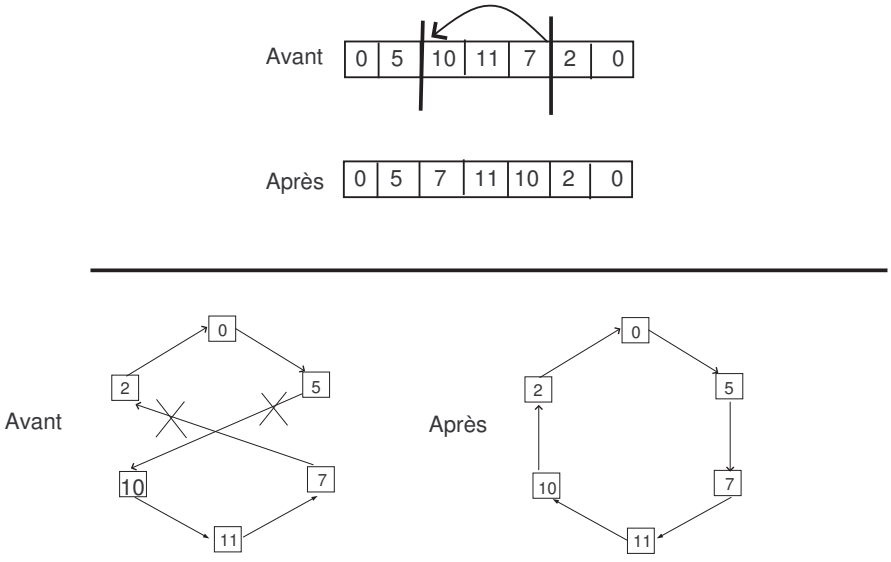


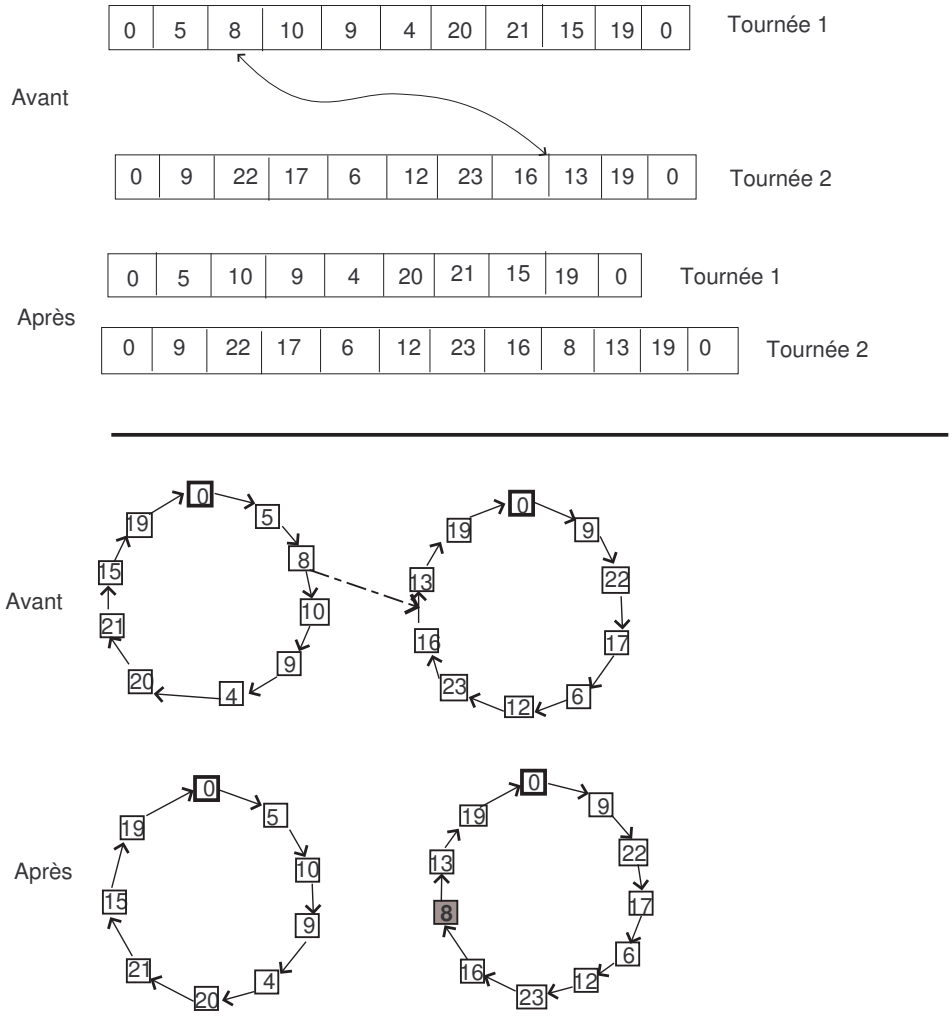
Figure 4.3 *Méthode d'amélioration : O*



Déplacement de demandes

Tout d'abord nous réalisons un déplacement d'une demande d'une tournée vers une autre. Ce mouvement est connu sous le nom de "String Relocation" [155] : il s'agit de déplacer une séquence de k nœuds d'une tournée vers une autre tournée. Dans notre problème nous aurons $k = 1$. Comme pour la méthode du 2-échange nous choisissons de ne déplacer qu'un nœud du fait de la présence des fenêtres de visite qui contraignent beaucoup la faisabilité d'une solution. Cette méthode sera appelée : D_o (pour Déplacement entre tournées). Dans la figure 4.4 nous schématisons le principe de cette méthode.

Figure 4.4 *Méthode d'amélioration : String Relocation (D_o)*

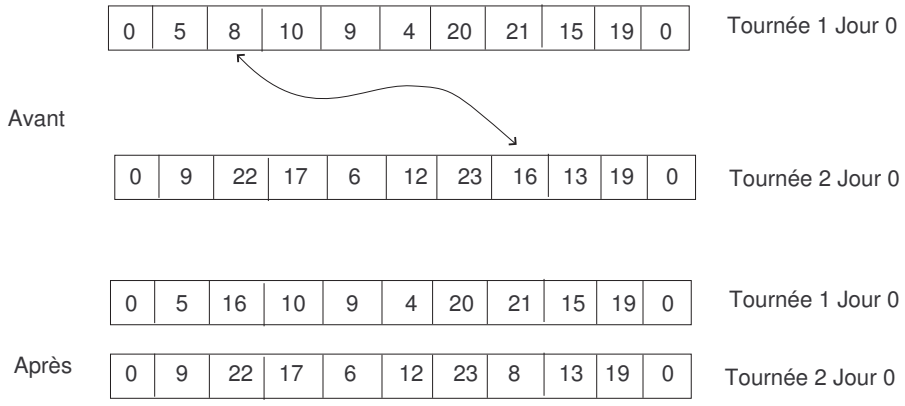


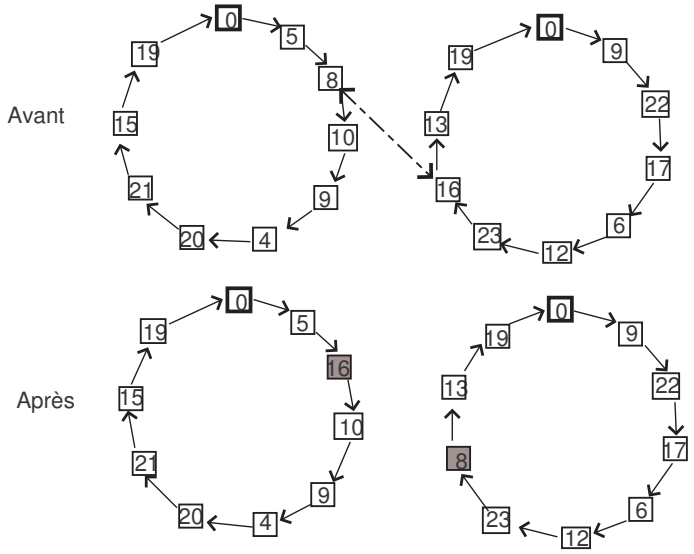
Échange de demandes

Ensuite nous réalisons l'échange de demandes entre tournées plus connu sous le nom de "String Exchange" [155]. Ce mouvement permet d'échanger une séquence de k nœuds d'une tournée avec une séquence de k nœuds d'une autre tournée. Nous prenons ici $k = 1$, de façon à être le plus rapide possible et afin d'éviter le plus possible les conflits dus aux fenêtres de visite. Cette méthode sera appelée : E_o (pour Échange entre tournées). Dans la figure 4.5 le principe de cette méthode y est schématisé.

Van Breedam [155] propose deux autres mouvements qui sont "String Cross" (échange de deux arcs appartenant à deux tournées différentes) et "String Mix" (le meilleur mouvement entre "String Relocation" et "String Exchange") mais nous ne les avons pas utilisés pour notre problème. En effet, avec "String Cross" les fins de deux tournées sont échangées, cela implique beaucoup de modifications pour chaque début de tournée (notamment le niveau de charge en produits du véhicule à son départ du dépôt) et implique beaucoup de situations non

Figure 4.5 *Méthode d'amélioration : String Exchange (E_o)*



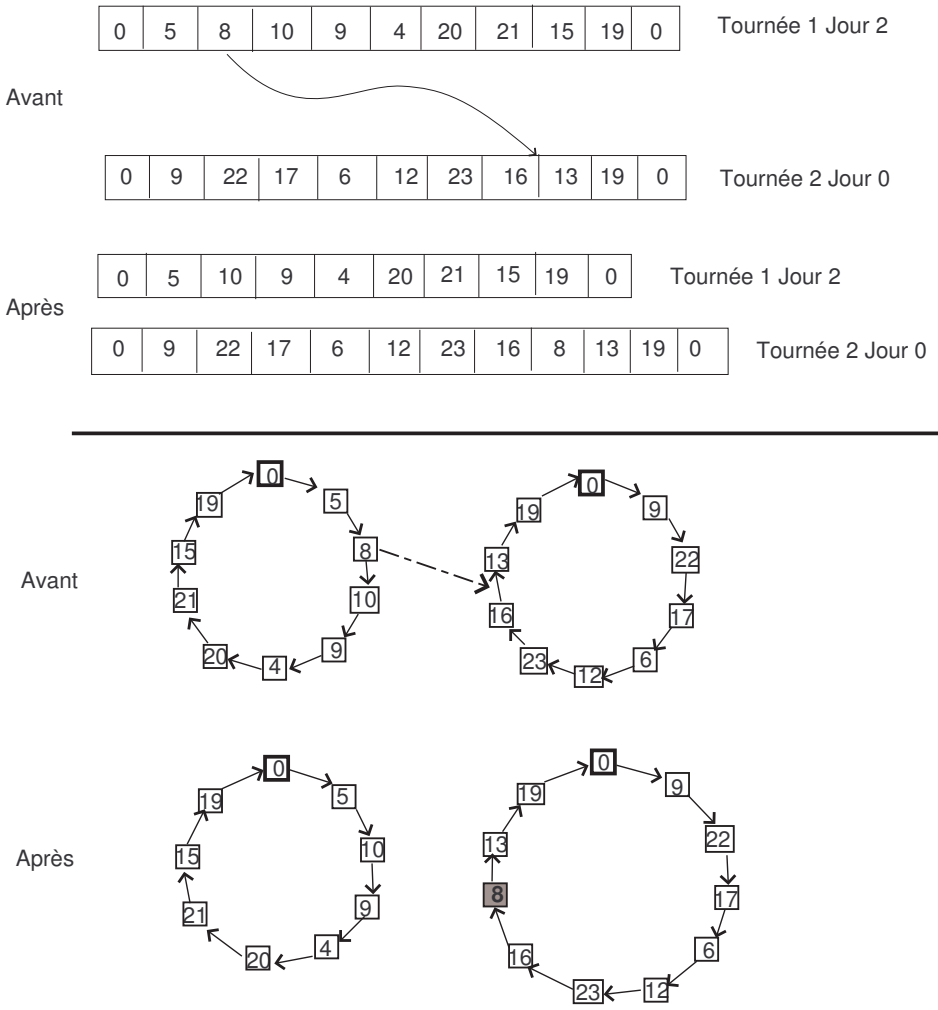


réalisables à cause des fenêtres de visite des sites et du respect de la capacité du véhicule en chaque site. Nous avons choisi également de ne pas utiliser "String Cross" car son temps d'exécution risque d'être long puisqu'il nécessite de connaître la solution obtenue par "String Relocation" et "String Exchange". Ces deux choix ont été faits dans un souci de rapidité d'exécution et d'efficacité de nos méthodes d'amélioration.

Déplacement de demande de livraison sur un jour antérieur

Nous utilisons également une troisième amélioration multi-tournées qui nous permet de traiter par avance les demandes de livraison d'une journée. En effet, ce voisinage permet de déplacer une demande d'une tournée vers une autre tournée effectuée un jour antérieur à son jour de demande ; ceci si le coût de stockage des produits livrés par avance est plus avantageux que le coût de routage de ces mêmes produits le jour demandé. Nous appellerons par la suite cette méthode *A* (pour Anticipation). Dans la figure 4.6 le principe de cette méthode y est expliqué.

Figure 4.6 *Méthode d'amélioration : anticipation de la demande : A*



4.3 Les différentes versions de tests

4.3.1 En juste à temps

Nous avons testé 6 combinaisons de méthodes d'amélioration. Comme il nous était impossible de tester l'ensemble des combinaisons possibles (720 combinaisons possibles), nous avons opté pour plusieurs stratégies : d'une part regrouper les catégories de mouvements – les échanges ensemble et les déplacements ensemble ; d'autre part nous avons voulu encadrer les mouvements mono-tournée par des mouvements multi-tournées (et

vice-versa). Ainsi, nous avons tout d'abord souhaité faire tous les échanges, suivi de tous les déplacements et enfin le 2-Opt, ensuite nous avons choisi de tester tous les déplacements suivi du 2-Opt et enfin des échanges et pour finir nous avons décidé de tester le 2-opt suivi de tous les échanges et enfin de tous les déplacements. Dans tous les cas sauf un ($D_oD_iOE_iE_o$), nous avons choisi de débuter nos améliorations par une amélioration mono-tournée suivie par toutes les améliorations multi-tournées pour enfin finir par une amélioration mono-tournée. Dans le cas $D_oD_iOE_iE_o$, nous choisissons de faire l'inverse, encadrer l'ensemble des améliorations mono-tournée par des améliorations multi-tournées. Le tableau 4.1 indique l'ordre d'utilisation des différents voisinages. Les noms donnés aux méthodes reprennent les abréviations données lors de la section 4.2.

Version	**Échange Mono** (E_i)	**Échange Multi** (E_o)	**Déplacement Multi** (D_o)	**Déplacement Mono** (D_i)	**2-Opt** (O)
$E_iE_oD_oD_iO$	1	2	3	4	5
$E_iE_oD_oD_iO^+$	1	2	3	4	5
$D_oD_iOE_iE_o$	4	5	1	2	3
$OE_iE_oD_oD_i$	2	3	4	5	1
$\overrightarrow{E_iE_oD_oD_iO}$	1	2	3	4	5
$\underline{\overrightarrow{E_iE_oD_oD_iO}}$	1	2	3	4	5

TAB. 4.1 – Combinaisons des méthodes d'amélioration testées pour la résolution de notre problème en juste à temps

Ces enchaînements de méthodes d'amélioration feront suite à chacune des deux méthodes de construction de solution expliquées dans la section 4.1, soit au total, 12 séries de tests. Ces enchaînements de méthodes peuvent s'apparenter à la méthode nommée VNS (Variable Neighbourhood Search) proposée par Hansen et Mladenović [85]. Cette méthode consiste à changer de voisinage afin d'explorer une région de plus en plus large autour de la solution courante.

Plusieurs versions sont présentées dans le tableau comme effectuant les mêmes méthodes d'amélioration et dans le même ordre ($E_iE_oD_oD_iO$, $E_iE_oD_oD_iO^+$, $\overrightarrow{E_iE_oD_oD_iO}$ et $\underline{\overrightarrow{E_iE_oD_oD_iO}}$). Pourtant les techniques d'amélioration diffèrent entre chaque version.

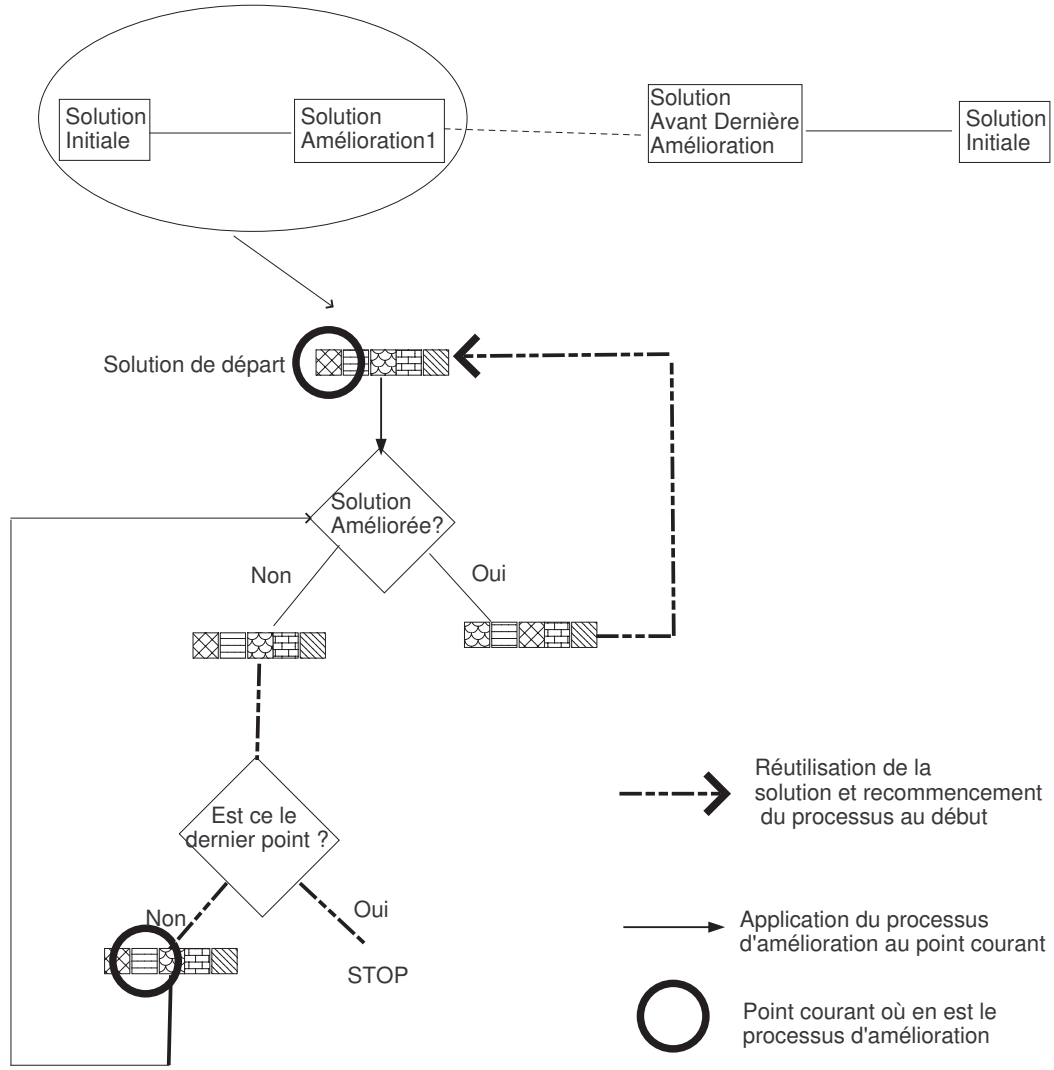
Tout d'abord comparons $E_iE_oD_oD_iO$ et $E_iE_oD_oD_iO^+$. $E_iE_oD_oD_iO$ (*cf.* Fig.4.7) enchaîne chaque méthode d'amélioration, la condition d'arrêt d'une méthode étant l'obtention d'un minimum local ; alors que dans $E_iE_oD_oD_iO^+$ (*cf.* Fig.4.8) la procédure de $E_iE_oD_oD_iO$ est réitérée tant qu'au moins une méthode d'amélioration améliore la solution.

Comparons maintenant les versions $E_iE_oD_oD_iO$, $\overrightarrow{E_iE_oD_oD_iO}$ et $\underline{\overrightarrow{E_iE_oD_oD_iO}}$. Comme il a été expliqué ci-dessus, dans la méthode $E_iE_oD_oD_iO$ on effectue méthode après méthode en ne changeant de méthode que lorsque le minimum local est atteint. Dans la méthode $\overrightarrow{E_iE_oD_oD_iO}$ (*cf.* Fig.4.9) et $\underline{\overrightarrow{E_iE_oD_oD_iO}}$ (*cf.* Fig.4.10) nous n'effectuons plus une méthode puis une autre mais à position de la solution réalisable nous effectuons toutes les méthodes d'amélioration dans l'ordre donné. La différence entre les versions $\overrightarrow{E_iE_oD_oD_iO}$ et $\underline{\overrightarrow{E_iE_oD_oD_iO}}$ est que $\overrightarrow{E_iE_oD_oD_iO}$ réalise la première amélioration qui lui est possible, alors que dans $\underline{\overrightarrow{E_iE_oD_oD_iO}}$ toutes les modifications possibles en un point sont testées avant de recommencer les améliorations.

Pour comprendre plus aisément les méthodes, nous donnons plusieurs schémas de déroulement. Chacun des schémas est accompagné par le pseudo-code de la méthode.

Dans la figure 4.7 nous expliquons le fonctionnement de la méthode $E_iE_oD_oD_iO$, son pseudo-code est donné dans l'algorithme 1. Avec la solution obtenue grâce à la construction, nous appliquons une première méthode d'amélioration jusqu'à l'obtention d'une amélioration, dès qu'une amélioration est trouvée la solution obtenue prend la place de la solution courante et on recommence le processus d'amélioration avec cette méthode, jusqu'à ce que cette méthode ne trouve plus d'amélioration. On passe alors à la méthode d'amélioration suivante.

Figure 4.7 *Méthode d'amélioration $E_iE_oD_oD_iO$*

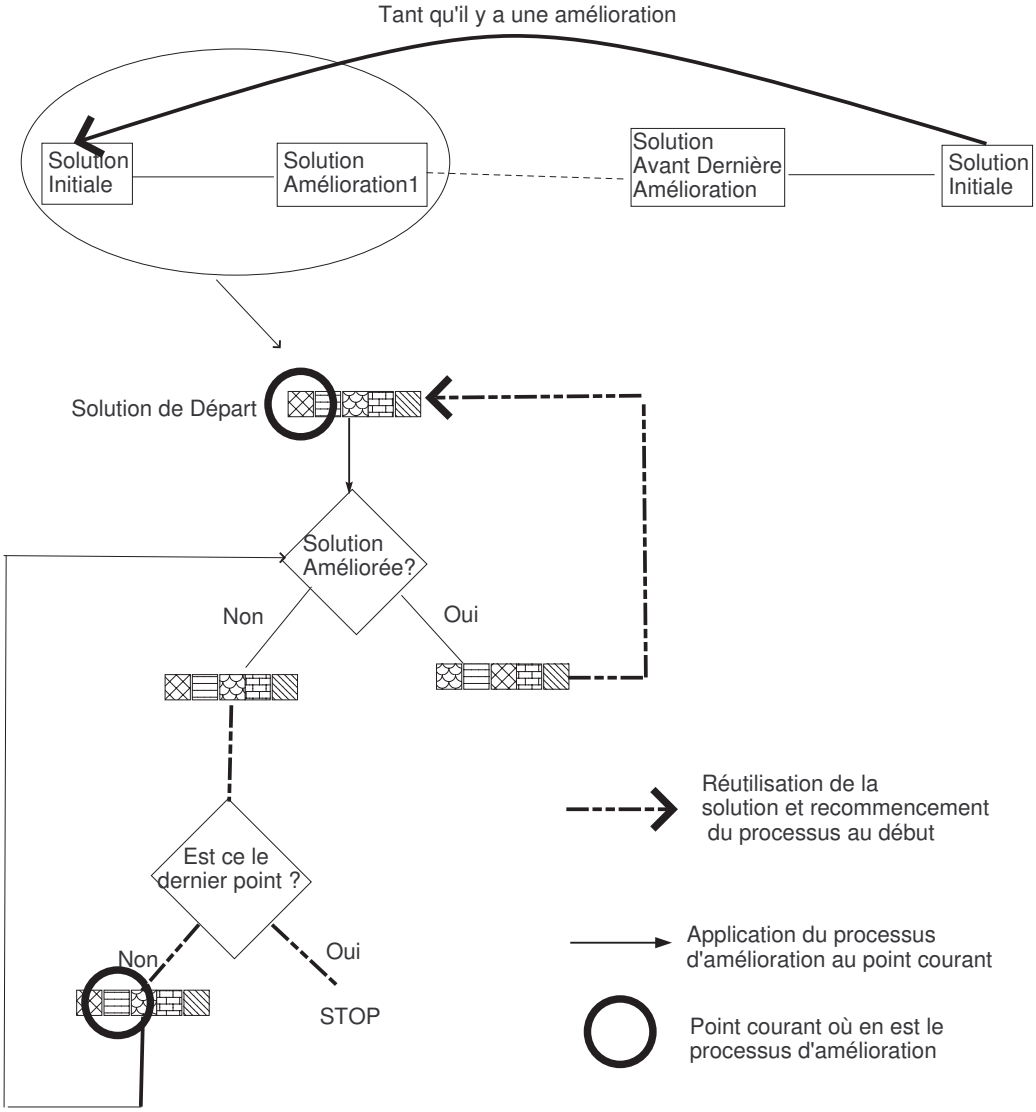


Algorithm 1 Recherche Locale utilisée par $E_iE_oD_oD_iO$

1: **for** i $\in$ [1,nb méthodes amélioration] **do**
2: **repeat**
3: Appliquer $MethodeAmelioration_i$ et s'arrêter dès la première amélioration
4: **until** (pas d'amélioration)
5: **end for**

Dans l'algorithme 1 la phase "Appliquer $MethodeAmelioration_i$" consiste à parcourir tous les points de la solution et de tester si une amélioration est possible avec la méthode i.

Figure 4.8 *Méthode d'amélioration $E_iE_oD_oD_iO^+$*



La figure 4.8 correspond à la méthode $E_iE_oD_oD_iO^+$ son pseudo-code est donné dans l'algorithme 4.3.1. Dans cette méthode le processus décrit par la figure 4.7 est réitéré tant qu'au moins une des méthodes d'amélioration améliore la solution.

La figure 4.9 montre le principe de la méthode nommée $\overrightarrow{E_iE_oD_oD_iO}$, son pseudo-code est donné dans l'algorithme 3. Dans cette méthode nous partons d'une solution obtenue grâce à la construction. Nous prenons le premier point de la première tournée et nous cherchons à lui appliquer toutes les méthodes d'amélioration, dès qu'une méthode améliore la solution, le mouvement est réalisé et le processus est alors stoppé. On réutilise la solution obtenue comme solution de départ. On recommence le principe tant qu'une amélioration au premier point de la tournée est réalisable. Lorsque plus aucune méthode n'améliore alors on passe au point suivant de la solution.

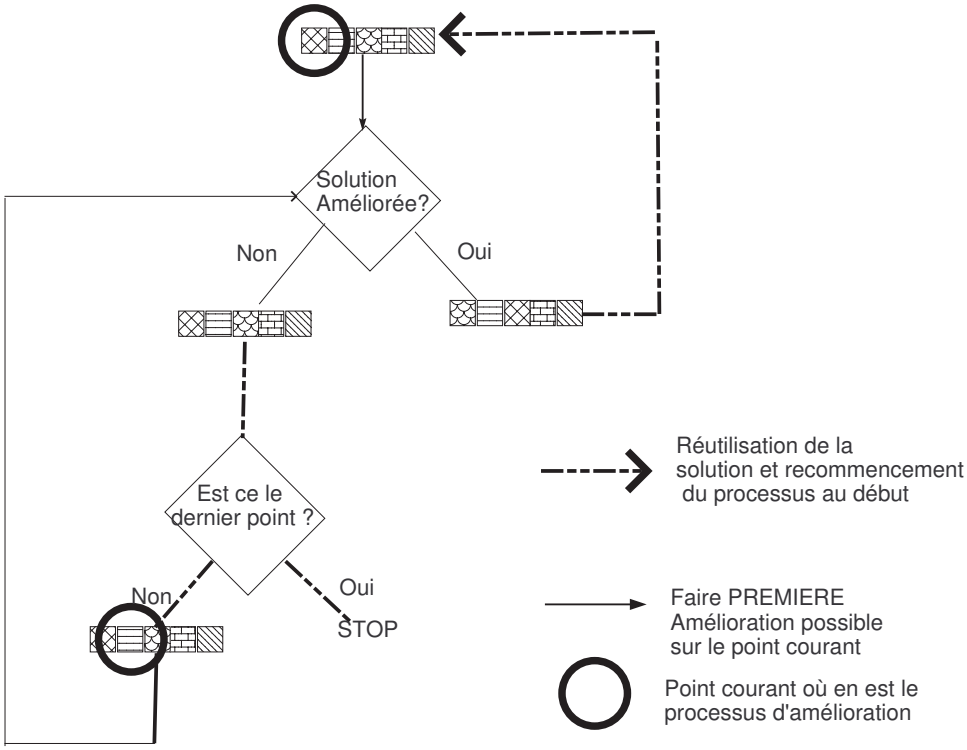
Dans l'algorithme 3, le principe "Appliquer les règles de la $MethodeAmelioration_i$" est différent de "$Appliquer\ la\ MethodeAmelioration_i$" puisqu'on ne parcourt pas tous les points de la solution comme avec le premier mais on applique juste le principe de la méthode en un point.

Enfin, dans la figure 4.10 nous trouvons le principe de la méthode $\underline{\overrightarrow{E_iE_oD_oD_iO}}$ son pseudo-code est donné dans l'algorithme 4. Dans cette méthode le principe est le même que dans celui expliqué dans la figure 4.9 sauf que l'on n'arrête pas le processus dès qu'une méthode améliore le processus mais on effectue toutes les amélio-

Algorithm 2 Recherche Locale utilisée par $E_i E_o D_o D_i O^+$

1: **repeat**
2: **for** i $\in$ [1,nb méthodes amélioration] **do**
3: **repeat**
4: Appliquer $MethodeAmelioration_i$ et s'arrêter dès la première amélioration
5: **until** (pas d'amélioration)
6: **end for**
7: **until** (pas d'amélioration)

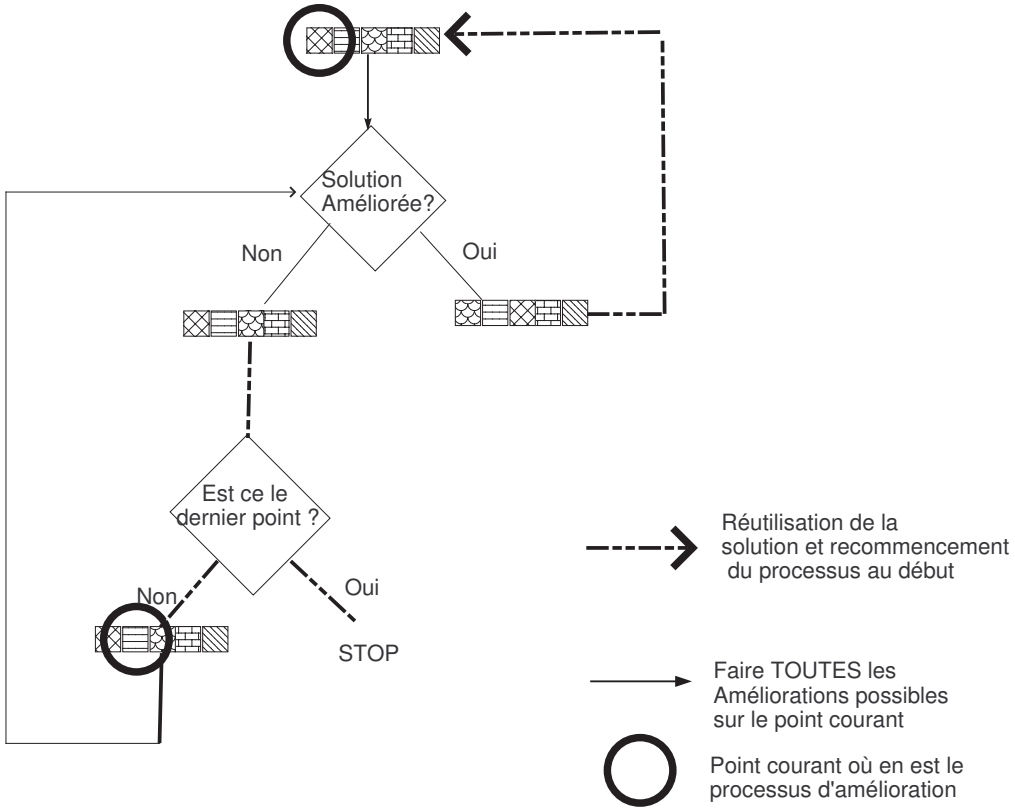
Figure 4.9 *Méthode d'amélioration* $\overrightarrow{E_i E_o D_o D_i O}$



Algorithm 3 Recherche Locale utilisée par $\overrightarrow{E_i E_o D_o D_i O}$

1: **for** chaque point de la solution courante **do**
2: **repeat**
3: Appliquer les règles de chaque $MethodeAmelioration_i$ jusqu'à amélioration
4: **until** (pas d'amélioration)
5: **end for**

Figure 4.10 *Méthode d'amélioration* $\overrightarrow{E_iE_oD_oD_iO}$



rations possibles.

Algorithm 4 Recherche Locale utilisée par $\overrightarrow{E_iE_oD_oD_iO}$

```
1: for chaque point de la solution courante do
2:     for i ∈ [1, nb méthodes amélioration] do
3:         Appliquer les règles de chaque MethodeAmelioration_i
4:     end for
5: end for
```

4.3.2 Avec gestion des stocks

Dans le tableau 4.2 nous donnons le descriptif des six méthodes utilisées dans les tests de la stratégie avec gestion des stocks.

Les explications concernant les versions sont les mêmes que dans le paragraphe 4.3.1.

4.4 Les résultats

Les résultats présentés dans cette partie sont obtenus avec les coûts suivants :
- coût de création d'une tournée : 2000 ;
- coût de routage : 10 ;
- coût de stockage : 10.

Les tests ont été effectués ici sur les 11 catégories déclinées en 3 sous-catégories (demande fixe et suivant les deux lois de Gauss). Les tests ont été réalisés sur un processeur Pentium *IV* à 2,8 GHz. Nous verrons dans la suite

Version	**Échange Intra** (E_i)	**Échange Inter** (E_o)	**Déplace Inter** (D_o)	**Déplace Intra** (D_i)	**2-Opt** (O)	**Avance** (A)
$E_iE_oD_oD_iOA$	1	2	3	4	5	6
$E_iE_oD_oD_iOA^+$	1	2	3	4	5	6
$D_oD_iOAE_iE_o$	5	6	1	2	3	4
$OE_iE_oAD_oD_i$	2	3	5	6	1	4
$\overrightarrow{E_iE_oD_oD_iOA}$	1	2	3	4	5	6
$\underrightarrow{E_iE_oD_oD_iOA}$	1	2	3	4	5	6

TAB. 4.2 – Combinaisons des méthodes d'amélioration testées pour la résolution de notre problème avec gestion des stocks

de ce document (*cf.* III) l'influence que peuvent avoir les différents coûts sur les résultats et sur les conclusions quant à l'avantage ou non de mettre en place une gestion des stocks.

4.4.1 Comparaison des deux méthodes de construction de solution

Version	**Nombre de meilleures solution**		**Tps Moyen Exécution (ms)**	**Écart moyen à la meilleure solution**	
MI	1759	96%	1703	9554	0,3 %
PMI	89	4%	1246	1260462	40 %

TAB. 4.3 – Comparaisons des résultats des deux méthodes de construction

Dans le tableau 4.3 nous donnons pour chaque méthode de construction le nombre de fois où elle obtient la meilleure solution, son temps moyen d'exécution et l'écart moyen qu'il obtient avec la meilleure solution. Nous constatons dans le tableau 4.3 que la méthode de construction MI seule est plus efficace que la méthode de construction PMI. En effet, dans 95 % des cas la méthode de construction MI donne de meilleurs résultats que PMI. La méthode de construction PMI semble plus rapide que MI mais donne vraiment de moins bons résultats. En effet, les écarts moyens avec la meilleure solution prouvent que lorsque MI ne trouve pas la meilleure solution il n'est pas aussi éloigné du meilleur résultat que peut l'être PMI.

4.4.2 Les tests des méthodes d'amélioration une à une

Nous avons choisi dans un premier temps, de tester l'efficacité des méthodes d'amélioration utilisées seules. Ainsi, nous testons les deux méthodes de construction suivies de chacune des méthodes d'amélioration.

Grâce aux tableaux 4.4 et 4.5 nous pouvons voir que les méthodes d'amélioration pouvant être considérées comme utiles (effectuant le plus de mouvements) pour notre problème semblent être : O (réalise dans 66 % des cas une amélioration), D_o (effectue dans 96% des cas une amélioration), E_o (effectue dans 62% des cas des améliorations) et A (effectue dans 95% des cas une amélioration). Toutes méthodes de construction confondues la méthode qui utilise D_i s'exécute en 1481 ms, E_i s'exécute en 1489 ms, celle qui utilise O s'exécute en 1484 ms, celle qui utilise D_o s'exécute en 1482 ms, celle qui utilise E_o s'exécute en 1493 ms et celle qui utilise A s'exécute en 1477 ms. On peut donc dire que les temps moyens d'exécution ne sont pas significativement différents pour chaque méthode d'amélioration.

Version	Nombre Solutions Améliorées		Nombre Solutions Min		Tps Exécution (ms)
$MI + D_i$	132	7%	27	1%	1709
$MI + E_i$	615	33%	35	2%	1711
$MI + O$	1061	57%	282	15%	1711
$MI + D_o$	1718	92 %	864	46%	1707
$MI + E_o$	657	35%	99	5%	1712
$MI + A$	1706	92%	513	27%	1706
$PMI + D_i$	375	20%	0	0%	1254
$PMI + E_i$	1301	70%	0	0%	1268
$PMI + O$	1358	73%	0	0%	1259
$PMI + D_o$	1843	99%	94	5%	1259
$PMI + E_o$	1644	88%	3	0,1%	1275
$PMI + A$	1839	99%	95	5%	1250

TAB. 4.4 – Tests des méthodes d'amélioration une à une en fonction des méthodes de construction

Version	Nombre Solutions Améliorées	Nombre Solutions Min	Coût Moyen
D_i	507	27	3956967
E_i	1916	35	3927139
O	2454	282	3947273
D_o	3561	958	3502519
E_o	2301	193	3777767
A	3545	608	3664671

TAB. 4.5 – Tests des méthodes d'amélioration une à une

4.4.3 Les tests sur les versions en juste à temps

Nous présentons ici les résultats obtenus sur l'ensemble de nos instances de tests grâce aux différentes combinaisons décrites dans le tableau 4.1. Le tableau 4.6, nous donne le nombre d'améliorations par rapport à la solution de construction, le nombre de solutions minimum obtenues, et le temps d'exécution pour chaque méthode de construction et améliorations sans avance. La méthode donnant les meilleurs résultats est $MI + E_iE_oD_oD_iO^+$ (62 % des cas). Nous constatons que dans 90% des cas il s'agit de la méthode de construction "Meilleure Insertion" qui offre les meilleurs résultats.

En regardant de plus près les résultats donnés pour chaque catégorie on se rend compte que plus les demandes des instances sont grandes plus la fréquence de meilleurs résultats donnés par cette méthode augmente (27% des instances de la catégorie A trouvent leur valeur minimum avec la méthode $MI + E_iE_oD_oD_iO^+$ alors que le pourcentage est de 66 % pour les instances de la catégorie C).

Les détails concernant les différents résultats obtenus selon les catégories d'instances de tests sont donnés en annexe de ce document (Annexe 3). En effet, dans le tableau 9.1 nous donnons les moyennes des coûts, obtenus par catégorie d'instances ainsi que la (ou les) version(s) donnant le meilleur résultat.

4.4.4 Les tests sur les versions avec gestion des stocks

Nous allons présenter ici les résultats obtenus grâce aux tests des différentes versions avec gestion des stocks présentées en section 4.2.

Le tableau 4.7, nous donne le nombre d'améliorations par rapport à la solution de construction, le nombre de solutions minimum obtenues grâce à la méthode, et le temps d'exécution pour chaque méthode de construction et améliorations avec gestion des stocks. On voit ici que la méthode donnant les meilleurs résultats est $E_iE_oD_oD_iOA^+$ qui donne dans 47% des cas la meilleure solution trouvée toutes méthodes de construction

Version	Nombre de Solutions Améliorées		Nombre de Solutions Min		Tps Exécution (ms)
$MI + E_iE_oD_oD_iO$	1815	98%	315	17%	1737
$MI + E_iE_oD_oD_iO^+$	1815	98%	1123	60%	1942
$MI + D_oD_iOE_iE_o$	1815	98%	333	18%	1744
$MI + OE_iE_oD_oD_i$	1815	98%	246	13%	1743
$\overrightarrow{MI + E_iE_oD_oD_iO}$	1814	98%	626	34%	2206
$\underline{\overrightarrow{MI + E_iE_oD_oD_iO}}$	1805	97%	220	11%	1729
$PMI + E_iE_oD_oD_iO$	1848	100%	9	0,4%	1315
$PMI + E_iE_oD_oD_iO^+$	1848	100%	163	8%	1527
$PMI + D_oD_iOE_iE_o$	1848	100%	33	2%	1306
$PMI + OE_iE_oD_oD_i$	1848	100%	15	0,8%	1308
$\overrightarrow{PMI + E_iE_oD_oD_iO}$	1848	100%	117	6%	2671
$\underline{\overrightarrow{PMI + E_iE_oD_oD_iO}}$	1848	100%	9	0,5%	1282

TAB. 4.6 – Résultats des différentes versions de résolution en juste à temps

Version	Nombre améliorations		Nombre meilleure solution		Tps Exécution
$MI + E_iE_oD_oD_iOA$	1821	98%	187	10%	1739
$MI + E_iE_oD_oD_iOA^+$	1821	98%	950	51%	1847
$MI + D_oD_iOAE_iE_o$	1821	98%	216	11%	1741
$MI + OE_iE_oAD_oD_i$	1821	98%	249	13%	1734
$\overrightarrow{MI + E_iE_oD_oD_iOA}$	1747	94%	161	8%	1992
$\underline{\overrightarrow{MI + E_iE_oD_oD_iOA}}$	1820	98%	271	14%	1736
$PMI + E_iE_oD_oD_iOA$	1848	100%	2	0,1%	1257
$PMI + E_iE_oD_oD_iOA^+$	1848	100%	192	10%	1440
$PMI + D_oD_iOAE_iE_o$	1848	100%	36	2%	1298
$PMI + OE_iE_oAD_oD_i$	1848	100%	47	3%	1300
$\overrightarrow{PMI + E_iE_oD_oD_iOA}$	1848	100%	49	3%	1911
$\underline{\overrightarrow{PMI + E_iE_oD_oD_iOA}}$	1848	100%	71	4%	1294

TAB. 4.7 – Résultats des différentes versions de résolution avec gestion des stocks

confondues.

Les meilleurs résultats sont encore une fois trouvés grâce à la méthode de construction MI (84 % des cas).

Les détails concernant les différents résultats obtenus selon les catégories d'instances de tests sont donnés en annexe de ce document (Annexe 3). En effet, dans le tableau 9.2 nous allons donner les moyennes des coûts obtenus par catégorie d'instances ainsi que la (ou les) version(s) donnant le meilleur résultat.

4.4.5 Comparaison des résultats en juste à temps et avec gestion des stocks

Version	Nombre de Meilleures Solutions
Juste à temps	423
Avec gestion des stocks	1666

TAB. 4.8 – Comparaison des résultats en juste à temps et avec gestion des stocks

Dans le tableau 4.8, nous comparons les résultats obtenus en juste à temps et avec gestion des stocks. Dans cette configuration de coûts, il est dans 90 % des cas plus avantageux de choisir d'anticiper les demandes, donc de faire de la gestion des stocks. Bien entendu si le coût de stockage devenait trop grand par rapport au coût de routage cette tendance s'inverserait. L'analyse de sensibilité par rapport aux différents coûts est réalisée ultérieurement dans ce rapport (*cf.* chapitre 8).

4.4.6 Les tests sur les versions restreintes

Dans cette section, nous allons tester les combinaisons de méthodes d'amélioration mais en ne gardant que les méthodes identifiées comme "utiles" dans la section 4.4.2. Ainsi, nous supprimons les méthodes E_i et D_i des différentes combinaisons, l'ordre d'exécution des méthodes reste le même.

Tests sur les versions restreintes en juste à temps

Dans le tableau 4.9, nous rappelons pour chaque méthode d'amélioration avec les méthodes faibles le nombre de solutions qui ont été améliorées par rapport à la solution obtenue par la méthode de construction, le nombre de solutions minimum trouvées parmi les 1848 tests, le temps d'exécution moyen et un indice par rapport au coût obtenu par la méthode de construction MI seule (Coût obtenu par la méthode MI représente l'indice 100). Nous fournissons les mêmes informations pour les méthodes d'amélioration sans les méthodes dites "faibles". Nous pouvons ainsi comparer les différentes versions testées et ce sous différents critères. Nous avons mis en évidence le coût moyen le plus faible. Nous constatons ici que toutes les versions de méthodes d'améliorations effectuent pour quasiment tous les fichiers de tests des améliorations face à la solution de construction obtenue.

Grâce aux tableaux : 4.9 et 4.10, nous pouvons constater que la version : "$MI + E_o D_o O^+$" donne le meilleur coût moyen sur toutes les versions testées en juste à temps. En effet, sans les méthodes dites faibles il s'agissait de la méthode "$MI + E_i E_o D_o D_i O^+$" qui donnait les meilleurs résultats, mais en enlevant les méthodes "E_i" et "D_i", la méthode "$MI + E_o D_o O^+$" trouve dans 1509 cas sur 1848 mieux ou autant que "$MI + E_i E_o D_o D_i O^+$".

Quant aux temps de calculs, ils sont quasi similaires pour l'obtention d'un optimum local au problème puisque pour les méthodes avec les "méthodes faibles" le temps de calcul moyen est de 1864 ms alors que sans le temps de calcul moyen est de 1806 ms.

Tests sur les versions restreintes avec gestion des stocks

Le tableau 4.11 présente les différents résultats obtenus avec les versions restreintes avec gestion des stocks. Comme pour les versions en juste à temps nous donnons ici les informations importantes concernant les différentes méthodes d'amélioration avec et sans les méthodes dites "faibles", de façons à pouvoir comparer selon plusieurs critères les méthodes. Nous constatons ici que toutes les versions de méthodes d'améliorations

Version	Nombre Solutions Améliorées		Nombre Solutions Min		Tps Exécution (ms)	Indice /MI
$MI + E_iE_oD_oD_iO$	1815	98%	314	17%	1737	93
$MI + E_iE_oD_oD_iO^+$	1815	98%	984	53%	1942	87
$MI + D_oD_iOE_iE_o$	1815	98%	318	17%	1744	93
$MI + OE_iE_oD_oD_i$	1815	98%	241	13%	1743	93
$\overrightarrow{MI + E_iE_oD_oD_iO}$	1814	98%	594	32%	2206	89
$\underline{\overrightarrow{MI + E_iE_oD_oD_iO}}$	1805	97%	215	11%	1729	91
$PMI + E_iE_oD_oD_iO$	1848	100%	9	0,4%	1315	109
$PMI + E_iE_oD_oD_iO^+$	1848	100%	145	7%	1527	99
$PMI + D_oD_iOE_iE_o$	1848	100%	29	2%	1306	109
$PMI + OE_iE_oD_oD_i$	1848	100%	15	0,8%	1308	110
$\overrightarrow{PMI + E_iE_oD_oD_iO}$	1848	100%	99	5%	2671	99
$\underline{\overrightarrow{PMI + E_iE_oD_oD_iO}}$	1848	100%	8	0,4%	1282	112
$MI + E_oD_oO$	1815	98%	312	17%	1737	93
$MI + E_oD_oO^+$	1815	98%	942	45%	1820	**87**
$MI + D_oOE_o$	1815	98%	314	17%	1740	93
$MI + OE_oD_o$	1815	98%	237	13%	1742	93
$\overrightarrow{MI + E_oD_oO}$	1814	98%	507	27%	1973	89
$\underline{\overrightarrow{MI + E_oD_oO}}$	1807	97%	146	8%	1935	97
$PMI + E_oD_oO$	1848	100%	2	0,1%	1300	110
$PMI + E_oD_oO^+$	1848	100%	114	6%	1414	99
$PMI + D_oOE_o$	1848	100%	17	0,9%	1299	110
$PMI + OE_oD_o$	1848	100%	10	0,5%	1301	111
$\overrightarrow{PMI + E_oD_oO}$	1848	100%	75	4%	2267	99
$\underline{\overrightarrow{PMI + E_oD_oO}}$	1848	100%	0	0%	1341	132

TAB. 4.9 – Résultats des différentes versions restreintes en juste à temps

Comparaison Versions	Nombre occurrences
$MI + E_iE_oD_oD_iO \geq MI + E_oD_oO$	1511
$MI + E_iE_oD_oD_iO^+ + \geq MI + E_oD_oO^+ +$	1509
$MI + D_oD_iOE_iE_o \geq MI + D_oOE_o$	1587
$MI + OE_iE_oD_oD_i \geq MI + OE_oD_o$	1568
$\overrightarrow{MI + E_iE_oD_oD_iO} \geq \overrightarrow{MI + E_oD_oO}$	1456
$\underline{\overrightarrow{MI + E_iE_oD_oD_iO}} \geq \underline{\overrightarrow{MI + E_oD_oO}}$	663
$PMI + E_iE_oD_oD_iO \geq PMI + E_oD_oO$	1105
$PMI + E_iE_oD_oD_iO + \geq PMI + E_oD_oO +$	1128
$PMI + D_oD_iOE_iE_o \geq PMI + D_oOE_o$	1116
$PMI + OE_iE_oD_oD_i \geq PMI + OE_oD_o$	1144
$\overrightarrow{PMI + E_iE_oD_oD_iO} \geq \overrightarrow{PMI + E_oD_oO}$	1160
$\underline{\overrightarrow{PMI + E_iE_oD_oD_iO}} \geq \underline{\overrightarrow{PMI + E_oD_oO}}$	208

TAB. 4.10 – Comparaison des résultats obtenus avec et sans les méthodes faibles identifiées

Version	Nombre Solutions Améliorées		Nombre Solutions Min		Tps Calcul (ms)	Indice / MI
$MI + E_iE_oD_oD_iOA$	1821	98%	171	9%	1739	90
$MI + E_iE_oD_oD_iOA^+$	1821	98%	598	32%	1847	**85**
$MI + D_oD_iOAE_iE_o$	1821	98%	184	9%	1741	90
$MI + OE_iE_oAD_oD_i$	1821	98%	216	11%	1734	89
$\overrightarrow{MI + E_iE_oD_oD_iOA}$	1747	94%	128	7%	1992	87
$\underline{\overrightarrow{MI + E_iE_oD_oD_iOA}}$	1820	98%	202	11%	1736	88
$PMI + E_iE_oD_oD_iOA$	1848	100%	2	0,1%	1257	96
$PMI + E_iE_oD_oD_iOA^+$	1848	100%	111	6%	1440	95
$PMI + D_oD_iOAE_iE_o$	1848	100%	19	1%	1298	104
$PMI + OE_iE_oAD_oD_i$	1848	100%	33	2%	1300	103
$\overrightarrow{PMI + E_iE_oD_oD_iOA}$	1848	100%	16	9%	1911	98
$\underline{\overrightarrow{PMI + E_iE_oD_oD_iOA}}$	1848	100%	31	2%	1294	97
$MI + E_oD_oOA$	1821	98%	184	10%	1838	90
$MI + E_oD_oOA^+$	1821	98%	570	31%	1923	85
$MI + D_oOAE_o$	1821	98%	181	10%	1751	90
$MI + OE_oAD_o$	1821	98%	196	11%	1733	89
$\overrightarrow{MI + E_oD_oOA}$	1821	98%	467	25%	2120	85
$\underline{\overrightarrow{MI + E_oD_oOA}}$	1838	99%	126	7%	1741	88
$PMI + E_oD_oOA$	1848	100%	0	0%	1357	105
$PMI + E_oD_oOA^+$	1848	100%	118	6%	1437	95
$PMI + D_oOAE_o$	1848	100%	16	0,8%	1293	104
$PMI + OE_oAD_o$	1848	100%	41	2%	1296	103
$\overrightarrow{PMI + E_oD_oOA}$	1848	100%	148	8%	2382	94
$\underline{\overrightarrow{PMI + E_oD_oOA}}$	1848	100%	17	0,9%	1292	98

TAB. 4.11 – Résultats des différentes versions restreintes avec gestion des stocks

effectuent pour quasiment tous les fichiers de tests des améliorations face à la solution de construction obtenue. On remarque cependant qu'il s'agit toujours des versions testées avec la méthode de construction MI qui offrent le plus de solutions minimum. La meilleure méthode sur le critère du coût moyen obtenu est la méthode "$MI + E_iE_oD_oD_iOA^+$". En effet elle donne le meilleur coût moyen et est plus rapide en temps de calcul que la version "$MI + E_oD_oOA^+$" qui est quasi similaire niveau performance.

Comparaison Versions	Nombre occurrences
$MI + E_iE_oD_oD_iOA \geq MI + E_oD_oOA$	1545
$MI + E_iE_oD_oD_iOA^+ \geq MI + E_oD_oOA^+$	1414
$MI + D_oD_iOAE_iE_o \geq MI + D_oOAE_o$	1500
$MI + OE_iE_oAD_oD_i \geq MI + OE_oAD_o$	1577
$\overrightarrow{MI + E_iE_oD_oD_iOA} \geq \overrightarrow{MI + E_oD_oOA}$	1462
$\underline{\overrightarrow{MI + E_iE_oD_oD_iOA}} \geq \underline{\overrightarrow{MI + E_oD_oOA}}$	1402
$PMI + E_iE_oD_oD_iOA \geq PMI + E_oD_oOA$	1134
$PMI + E_iE_oD_oD_iOA^+ \geq PMI + E_oD_oOA^+$	1143
$PMI + D_oD_iOAE_iE_o \geq PMI + D_oOAE_o$	965
$PMI + OE_iE_oAD_oD_i \geq PMI + OE_oAD_o$	1178
$\overrightarrow{PMI + E_iE_oD_oD_iOA} \geq \overrightarrow{PMI + E_oD_oOA}$	1412
$\underline{\overrightarrow{PMI + E_iE_oD_oD_iOA}} \geq \underline{\overrightarrow{PMI + E_oD_oOA}}$	988

TAB. 4.12 – Comparaison des différentes versions avec gestion des stocks

Dans le tableau 4.12 nous comparons le nombre de fois où la version avec les méthodes faibles est supérieure ou égale à la version sans les méthodes faibles, autrement dit, le nombre de fois où la version sans les méthodes faibles est autant voire plus efficace que celles avec. Nous constatons ici que globalement il est autant voire plus efficace, en terme de nombre de solutions meilleures, d'utiliser une version sans les méthodes faibles. Pour autant, on voit dans le tableau 4.11, qu'en terme de coût moyen de solution, la meilleure solution est donnée par une méthode comportant les méthodes faibles. Comme notre objectif est avant tout de minimiser les coûts globaux d'exploitation nous allons donc privilégier les résultats obtenus sur les coûts moyens et donc préférer la méthode : "$MI + E_iE_oD_oD_iOA^+$".

4.5 Conclusions et Perspectives

La meilleure méthode de construction identifiée ici est "Meilleure Insertion". Dans tous les cas nous pouvons dire que la meilleure méthode d'amélioration de solution est celle proposée comportant un "+". Certains voisinages ont pu être identifiés comme "faibles" lorsqu'ils sont utilisés seuls mais nous avons pu voir dans la version avec gestion des stocks qu'ils apportent quand même une diversité de solutions qui permet d'atteindre de meilleures solutions. C'est pour cette raison que par la suite nous utiliserons comme méthode heuristique "$MI + E_iE_oD_oD_iOA^+$" pour des travaux avec gestion des stocks et "$MI + E_oD_oO^+$" pour des travaux en juste à temps.

Nous avons ici décidé de traiter l'aspect "avance de la demande" et par conséquent gestion des stocks par une recherche de voisinage. Une autre méthode aurait été de traiter cette facette de notre étude directement dans la partie construction de solution. Le temps de calcul aurait été par conséquent lourdement augmenté. En effet, dans notre construction de solution, on cherche à placer une demande dans les tournées réalisées le jour de la demande alors qu'il faudrait, dans cette nouvelle méthode, tester l'insertion dans toutes les tournées réalisées dans un jour inférieur ou égal à celui considéré.

Nous avons dans ce chapitre testé différentes méthodes de construction et améliorations afin de pouvoir identifier la méthode la plus adaptée à notre problème. Les deux méthodes mises en évidence par les différents tests effectués vont être par la suite utilisées pour la résolution de notre problème grâce à la métaheuristique GRASP (Greedy Randomized Adaptive Search Procedure).

Une première perspective concernant les méthodes de constructions et améliorations serait l'emploi de techniques de programmation par contraintes au sein de la méthode de construction. En effet, une fois toutes les

demandes planifiées et donc attribuées à une tournée, nous chercherions pour chaque tournée quel est l'ordre de visite optimal minimisant le coût de transport tout en respectant les fenêtres de visite des sites et la capacité des véhicules.

Une seconde perspective résiderait dans l'emploi de méthodes d'apprentissage pour tenter "d'apprendre" la meilleure combinaison des techniques d'amélioration présentées.

Une troisième perspective concerne une éventuelle utilisation industrielle. Il faudrait alors mutualiser l'ensemble des heuristiques testées dans ce chapitre afin de ne donner que la meilleure solution à un utilisateur qui ne chercherait pas le meilleur résultat en moyenne mais plutôt à toujours avoir la meilleure solution. En effet il est rare que toutes les heuristiques échouent sur la même instance. De plus la rapidité des heuristiques mises place nous permet la multiplication des tests. On a donc intérêt à exécuter toutes les heuristiques disponibles et à conserver la meilleure solution.

Chapitre 5

Résolution par métaheuristique : GRASP

Dans ce chapitre, nous allons expliquer comment nous avons résolu notre problème par le biais d'une métaheuristique appelée : GRASP (Greedy Randomized Adaptive Search Procedure). L'utilisation de cette méthode de résolution est justifiée par les bons résultats que nous avons obtenus avec cette méthode sur un problème d'IRP (Inventory Routing Problem) [80]. Cette méthode de résolution sera dans un premier temps utilisée avec des techniques purement empruntées à la recherche opérationnelle, puis nous hybriderons cette méthode grâce à des techniques de programmation par contraintes.

5.1 Le GRASP (Greedy Randomized Adaptive Search Procedure) :

5.1.1 Le principe de la méthode de résolution

Nous considérons un problème d'optimisation combinatoire défini par :
- un ensemble fini $E = \{1, \ldots, n\}$ d'éléments ;
- un ensemble de solutions réalisables $F \subseteqq 2^E$;
- une fonction économique $f : 2^{|E|} \rightarrow \mathbb{R}$ telle que $f(S) = \sum_{e \in S} c(e) \quad \forall S \in 2^{|E|}$,
  où c(e) est le coût associé à la participation de l'élément e dans la solution S.

Ici, dans un cas de minimisation, nous recherchons une solution optimale $S^* \in F$ telle que :

$$\forall\, S \in F \quad f(S^*) \leq f(S)$$

La métaheuristique GRASP est un processus itératif. Chaque itération du GRASP comprend deux étapes (*cf.* algorithme 5) :
- la construction d'une solution ;
- la recherche de voisinage.

Une solution réalisable est construite lors de chaque itération du GRASP à la fin de l'étape de construction (*cf.* algorithme 6). Pour cela, la construction débute par une solution vide, chaque élément sera inséré un à un dans la solution. À chaque itération de la méthode de construction, le choix du prochain élément à incorporer dans la solution en construction est déterminé par l'évaluation de tous les éléments qui ne font pas encore partie de la solution partielle, selon une fonction d'évaluation. Cette méthode de construction de solution utilisée dans le GRASP est appelée gloutonne car à chaque itération on prolonge le chemin sur le plus "proche" (au sens des coûts) sommet non encore visité. Cette fonction représente l'augmentation immédiate de coût apportée par l'incorporation de cet élément dans la solution en construction. L'évaluation des éléments selon cette fonction mène à la création d'une liste restreinte de candidats (nommée RCL) formée par les meilleurs éléments suivant le critère considéré, c'est-à-dire ceux dont l'incorporation à la solution suivent le plus le critère (il s'agit de l'aspect glouton – greedy – de la méthode). L'élément sélectionné et incorporé à la solution partielle est tiré au hasard parmi les meilleurs dans la liste restreinte de candidats ; il n'est pas forcément le meilleur (il s'agit ici de l'aspect probabiliste – randomized – de la méthode). Ce critère de choix permet à différentes solutions d'être construites au cours des itérations de la méthode, sans toutefois trop compromettre leur qualité. L'amélioration associée à chaque élément non encore placé dans la solution réalisable est évaluée de nouveau à la fin de chaque itération de manière à prendre en considération les modifications de la structure de la solution en construction dues à la

sélection de l'élément précédent (il s'agit de l'aspect adaptatif – adaptive).

Les solutions réalisables obtenues à la fin de l'étape de construction ne sont pas forcément optimales, même par rapport à des voisinages définis de manière assez simple. L'application d'une méthode de recherche de voisinage (*cf.* algorithme 7) permet souvent de les améliorer. Ce type de méthode commence à partir d'une solution réalisable, qui est successivement remplacée par une solution meilleure appartenant à son voisinage ; un voisinage N associe à chaque solution S un sous-ensemble $N(S)$ de solutions. La solution S est un optimum local par rapport au voisinage $N(S)$ s'il n'existe pas de solution strictement meilleure que S dans $N(S)$. L'algorithme s'arrête lorsqu'un minimum local est atteint, c'est-à-dire lorsque les solutions voisines de la solution courante lui sont de qualité inférieure, en termes de coût.

L'efficacité d'une méthode de recherche de voisinage dépend de plusieurs facteurs, tels que la structure de voisinage, la technique de recherche des solutions dans le voisinage, le calcul rapide des coûts des solutions voisines et la solution de départ. L'étape de construction joue donc un rôle important par rapport à ce dernier aspect, en produisant de bonnes solutions de départ pour la recherche de voisinage. Des voisinages simples, tels que les 2-échanges sont souvent utilisés.

L'ensemble de ces deux étapes (construction et recherche de voisinage) est réitéré un certain nombre de fois (paramètre de l'algorithme). La meilleure solution obtenue parmi ses itérations sera retenue.

La méthode GRASP peut-être considérée comme une métaheuristique qui combine les bonnes caractéristiques des algorithmes gloutons (convergence rapide et bonnes solutions) avec celles des méthodes probabilistes multi départs (diversité des solutions). Cette méthode permet de proposer plusieurs solutions pour un même problème, ce point là est un atout dans les problèmes de construction de tournées où souvent la solution proposée est amenée à être rectifiée à cause de problèmes liés à des travaux sur les routes, à des accidents ... Dans ces cas là, une alternative à la meilleure solution peut être utilisée en sélectionnant une des autres tournées générées par l'algorithme.

5.1.2 La littérature

La méthode GRASP a été introduite pour la première fois par Feo et Resende [61] en 1989. De nombreux documents explicatifs ont pu être trouvés dans la littérature, citons notamment les travaux de Pitsoulis et Resende [123] qui décrivent les améliorations du GRASP, la parallélisation de la méthode et étudient les différents travaux réalisés grâce au GRASP pour résoudre des problèmes de logique, de localisation et d'allocation. Nous pouvons aussi citer les travaux de Ribeiro [128], qui étudie les différents travaux réalisés sur le GRASP et leurs applications ainsi que la parallélisation de la méthode et son hybridation. La méthode a déjà fait ses preuves dans des problèmes de construction de tournées, ainsi Kontoravdis et Bard 1995 [99], l'utilisent pour résoudre un problème de constructions de tournées avec fenêtres de temps où le but est de minimiser la taille de la flotte utilisée, la phase de recherche locale est réalisée dans cette étude toutes les 5 itérations de la première phase (phase de construction de solution) sur la meilleure solution trouvée. Citons également nos travaux (Grellier *et al.* 2004 [80]). Nous utilisons cette méthode pour résoudre un problème de construction de tournées avec gestion des stocks, une méthode de meilleure insertion pour la construction de solution est utilisée. Nous comparons la réalisation de la seconde phase au bout de 3 ou 5 itérations de la première phase. La méthode GRASP a également été employée pour d'autres domaines que celui de la construction de tournées : problème de capacité d'infrastructure ferroviaire (Delorme *et al.* [44]), problème de localisation (Bautista et Pereira [9]), problème de partitionnement (Areibi et Vannelli [3]), problème d'ordonnancement (Feo *et al.* [62])... Presque toutes les références données concernant le GRASP peuvent être retrouvées dans la bibliographie annotée faite par Festa et Resende [63] qui référence une grande partie des articles ayant trait au GRASP.

Un lien peut être fait entre la méthode GRASP et l'optimisation par colonies de fourmis [53] (ACO : Ant Colony Optimization). En effet un algorithme ACO est une forme de GRASP où la phase de construction de solution évolue au cours de l'exécution. Le caractère adaptatif est plus développé dans un algorithme ACO que pour le GRASP.

5.1.3 Les algorithmes

Les entrées des différents algorithmes sont :
- Critère : le critère de construction de la liste de candidats ;
- Max_Itération : le nombre total d'itérations (construction + recherche de voisinage) ;

– Racine : la racine pour le générateur de nombres aléatoires.

Algorithm 5 Procédure GRASP (Critère, Max_Itération)

```
Entrée des données ;
for k=1,..., Max_Itération do
   Solution ← Construction-Gloutonne-Randomisee(Critère)
   Solution ← Recherche-De-Voisinage (Solution)
   Mise-A-Jour-Solution(Solution, Meilleure-Solution)
end for
Retourner Meilleure-Solution
```

La procédure 5 renvoie la meilleure solution trouvée grâce à la méthode GRASP.

Algorithm 6 Procédure Construction-Gloutonne-Randomisee (Critère)

```
Solution = ∅
Évaluer la contribution de chaque élément selon Critère
while Solution n'est pas complète do
   Créer la liste restreinte de candidats RCL
   Choisir un élément s de RCL suivant une loi de probabilité
   Solution ← Solution ∪ {s}
   Évaluer à nouveau la contribution de chaque élément
end while
Retourner Solution
```

La procédure 6 renvoie la solution obtenue grâce à la méthode de construction.

Algorithm 7 Procédure Recherche-De-Voisinage(Solution)

```
while Solution n'est pas optimum local do
   Obtenir une Solution Voisine meilleure que Solution dans son voisinage N(Solution)
   Solution ← Voisine
end while
Retourner Solution
```

La procédure 7 renvoie la solution obtenue suite à la méthode de recherche de voisinage.

5.2 Le GRASP appliqué à notre problème

Pour la résolution de notre problème grâce à la métaheuristique GRASP nous choisissons deux types de résolution. Tout d'abord, la résolution du problème dite "classique", qui utilise des techniques que nous avons utilisées pour la résolution du problème par des méthodes de construction et améliorations (chapitre 4). Ensuite, nous utilisons la résolution dite "hybride" où la partie recherche locale de la métaheuristique est réalisée grâce à des techniques de programmation par contraintes.

5.2.1 La résolution "classique"

Grâce à notre étude sur les méthodes de construction et améliorations (*cf.* chapitre 4) nous avons choisi la méthode de construction et celles d'améliorations qui nous paraissaient les meilleures. La méthode de construction qui a été identifiée comme la plus probante est "Meilleure Insertion", nous l'utilisons donc. Meilleure Insertion est basée sur le principe suivant : pour chaque demande nous cherchons quelle est la tournée dans laquelle il faut l'insérer, et au sein de cette tournée nous cherchons à quel endroit nous devons l'insérer (*c.-à-d.* entre quels sites déjà présents dans la tournée) et quelle part de la demande (quantité) est prise en charge par cette tournée, de

façon à ce que l'on ait le meilleur coût (coût minimum de routage). Nous définissons donc pour chaque demande le meilleur triplet : tournée, placement dans la tournée et quantité qui offre le coût minimum d'insertion de cette demande.

Nous partons donc d'une solution vide (où toutes les tournées effectuent le trajet dépôt→ dépôt). Chaque élément non planifié dans la solution est alors évalué par rapport à la fonction d'évaluation "Meilleure Insertion" (on ne retient que la meilleure de toutes les évaluations pour chaque élément). Une liste de candidats à l'insertion est alors construite. Ces candidats correspondent aux sites ayant les plus petits coûts d'insertion dans la solution. Nous verrons par la suite que nous utiliserons successivement une liste de candidats de taille 3 et une liste de taille 5 afin de comparer les résultats obtenus. Le candidat inséré dans la solution est alors choisi aléatoirement parmi les sites placés dans la liste des candidats. L'insertion de ce candidat dans la solution courante est alors effectuée et nous réitérons le processus jusqu'à ce que plus aucune demande ne soit à planifier. Le processus de construction de solutions est itéré n fois (paramètre de la résolution). Suite à un certain nombre d'itérations de la phase de construction nous effectuons une recherche locale sur la meilleure solution obtenue au cours de ces étapes de construction. La méthode de recherche locale diffère selon si l'on décide de pouvoir livrer en avance (ce qui implique un coût de stockage) ou non les demandes des livraisons des sites.

La recherche locale pour la résolution du problème en juste à temps

La meilleure recherche locale qui a été identifiée lors de notre étude sur les méthodes de construction et améliorations pour une résolution du problème en juste à temps des livraisons est nommée : "$E_o D_o O^+$". Cette méthode est composée des heuristiques d'amélioration suivantes :

- E_o : échange de demandes entre tournées plus connu sous le nom de "String Exchange". Ce mouvement permet d'échanger une séquence de k nœuds d'une tournée avec une séquence de k nœuds d'une autre tournée. Nous prenons ici $k = 1$.
- D_o : déplacement d'une demande d'une tournée vers une autre. Ce mouvement est connu sous le nom de "String Relocation" : il s'agit de déplacer une séquence de k nœuds d'une tournée vers une autre tournée. Dans notre problème nous aurons $k = 1$. Comme pour la méthode du 2-échange nous choisissons de ne déplacer qu'un nœud du fait de la présence des fenêtres de visite qui contraignent beaucoup la faisabilité d'une solution.
- O : le 2-opt, qui consiste à échanger 2 arcs d'une tournée puis à re-connecter les morceaux de cette tournée.

On enchaîne chaque méthode d'amélioration (la condition d'arrêt d'une méthode étant l'obtention d'un minimum local) et l'enchaînement de ces méthodes est réitéré tant qu'au moins une méthode d'amélioration améliore la solution courante.

La recherche locale pour la résolution du problème avec gestion des stocks

La meilleure recherche locale qui a été identifiée lors de notre étude sur les méthodes de construction et améliorations pour une résolution du problème avec gestion des stocks des livraisons est nommée : "$E_i E_o D_o D_i O A^+$". Cette méthode est composée des heuristiques d'amélioration suivantes :

- E_i : le 2-échange de nœuds au sein d'une même tournée. Cette méthode consiste à échanger deux nœuds qui sont planifiés au sein d'une même tournée.
- E_o : *cf.* explications données dans le paragraphe au dessus.
- D_o : *cf.* explications données dans le paragraphe au dessus.
- D_i : la méthode Or-opt [118] qui consiste à déplacer une séquence de 3, 2 ou 1 nœuds consécutifs au sein d'une tournée. Pour notre étude nous avons choisi de déplacer des séquences de un nœud. Nous choisissons de ne pas déplacer plus de nœuds car nos sites ont des fenêtres de visite donc en déplaçant plus de nœuds nous risquerions de nous trouver très fréquemment avec des solutions non réalisables.
- O : *cf.* explications données dans le paragraphe au dessus.
- A : amélioration multi-tournées qui nous permet de traiter par avance les demandes de livraison d'une journée. En effet, ce voisinage permet de déplacer une demande d'une tournée vers une autre tournée effectuée un jour antérieur à son jour de demande ; ceci si le coût de stockage des produits livrés par avance est plus avantageux que le coût de routage de ces mêmes produits le jour demandé.

5.2.2 La résolution hybride

La résolution hybride utilise la même méthode de construction que celle décrite pour les méthodes dites "classiques" (*c.-à-d.* meilleure insertion). C'est lors de la phase de recherche locale que nous introduisons des techniques de programmation par contraintes. Nous allons utiliser la technique LNS (Large Neighbourhood Search) introduite par Shaw en 1998 [137]. Cette méthode consiste à sélectionner un certain nombre de sites de la solution, à relaxer leur position et à les réinsérer en utilisant des techniques de programmation par contraintes. Nous partons donc d'une solution (obtenue sur le même principe que décrit en 5.2.1) nous choisissons aléatoirement deux positions de site que nous relaxons. Ces deux sites deviennent donc des variables.

Le choix des sites à retirer

Aléatoirement Dans cette version, la sélection des sites à retirer dans la solution réalisable, obtenue lors de la première phase du GRASP, est réalisée de façon aléatoire.

Plus forte contribution Dans cette deuxième version de sélection des sites, celle-ci est réalisée en calculant la contribution de chaque site dans la solution. En effet, pour chaque site de la solution nous calculons sa contribution qui représente l'augmentation du coût de la solution due au fait de la visite du site. Nous pouvons noter c_i la contribution du site i à la solution, p_i le prédécesseur de i dans la solution et s_i son successeur. d_{ij} représente la distance séparant les sites i et j. CC correspond au coût de création d'une tournée (coût fixe), CR le coût de routage (coût variable) et x_i est égal à 1 si le site i est seul dans la tournée (*i.e.* si son retrait entraîne une tournée vide) et 0 sinon. Nous obtenons ainsi : $c_i = CR \times (d_{p_i,i} + d_{i,s_i} - d_{p_i,s_i}) + CC \times x_i$. Le site ayant la plus forte contribution est alors enlevé, il s'agit en effet du site qui apporte la plus forte augmentation au niveau du coût à la solution. Le calcul des contributions est alors réitéré pour prendre en compte la modification effectuée par le retrait du premier site, puis le second site (celui ayant la plus forte contribution suite à la deuxième évaluation) est enlevé.

Similitude des nœuds Dans la troisième version de sélection des sites à enlever, nous choisissons ceux ayant des similitudes communes. En effet, nous cherchons parmi tous les couples de sites de la solution précédente, celui ayant une similitude la plus forte. Nous notons $Simi_{ij}$ la valeur de la similitude des sites i et j. Deux sites ont une similitude élevée si la différence entre leur instant de visite dans la solution réalisable est faible. Nous cherchons ainsi à retirer deux sites de deux tournées différentes de façons à pouvoir les intervertir. Soit t_i l'instant de visite du site i, nous avons alors $Simi_{ij} = |t_i - t_j|$. Nous sélectionnons alors le couple ayant la plus petite similitude.

Le modèle

Deux principales contraintes sont à vérifier dans ce modèle : le respect des fenêtres de visite des sites et le respect de la capacité des véhicules.

La contrainte des fenêtres de visite consiste à s'assurer que chaque site est visité dans sa période d'ouverture. Pour cela nous introduisons pour chaque site une variable nommée "Instant de Visite" dont le domaine de définition est la fenêtre de visite du site. Le dépôt a un instant de visite fixe qui est égal à 0. Pour chaque site, on assure la cohérence des instants de visite à partir de son prédécesseur dans la tournée. En effet, la variable instant de visite du $site_i$ sera toujours supérieure ou égale à la variable instant de visite du prédécesseur du $site_i$ plus le temps de service du prédécesseur du $site_i$ plus la distance séparant les deux sites (on part du principe qu'il faut une unité de temps pour parcourir une unité de distance).

Soient $t_{i,r}$ l'instant de visite du site i dans la tournée r, p_{ir} le prédécesseur de i dans la tournée r, d_{ij} est la distance qui sépare i de j et ts_i est le temps de service du site i. Nous avons alors :

$$\forall\, i \quad \forall\, r \quad t_{ir} \geq t_{p_{ir}r} + d_{p_{ir}i} + ts_{p_{ir}} \tag{5.1}$$

La contrainte de respect de la capacité des véhicules consiste à s'assurer qu'en tout point de la tournée le véhicule ne dépasse jamais sa capacité de stockage et ne livre pas plus que ce qu'il a dans son stock. Pour cela nous mettons en place quatre variables pour chaque demande i :

– la variable AC_i : représente la capacité libre du véhicule avant son arrivée en i ;

- la variable DC_i : représente la capacité libre du véhicule après son passage en i ;
- la variable MAC_i : représente le minimum des capacités libres du véhicule à son arrivée chez tous les prédécesseurs de i, i compris. Si P_i représente l'ensemble des sites visités avant le site i alors $MAC_i = \min_{P_i \cup i}(AC_i)$;
- la variable MDC_i : représente le minimum des capacités libres du véhicule après son passage chez tous les successeurs de i, i compris. Si S_i représente l'ensemble des sites visités après le site i alors $MDC_i = \min_{S_i \cup i}(DC_i)$.

Les quatre variables ont pour domaine $[0, capaciteVehicule]$. Une cinquième variable nommée : $PdtLiv$ représente l'ensemble des produits qui sont livrés par la tournée. La variable AC du dépôt est égale à $capaciteVehicule$, la variable DC du dépôt est égale à $capaciteVehicule - PdtLiv$. Chaque variable DC est égale à la variable AC de son successeur. Pour vérifier si une insertion d'un site k de livraison entre le site i et j est possible (uniquement vis à vis de la contrainte de capacité des véhicules) il suffit de vérifier que la quantité $qtePdt_k$ de produits livrée au site k est telle que $qtePdt_k \leq MAC_i$. De même, pour vérifier si une insertion d'un site k de collecte entre le site i et j est possible il suffit de vérifier que la quantité $qtePdt_k$ de produits collectée au site k respecte $qtePdt_k \leq MDC_j$.

Les autres contraintes qui doivent être vérifiées sont :

- La correspondance entre prédecesseur et successeur d'un site. Notons $Prec_i$ le prédécesseur de i et $Succ_i$ le successeur de i, il faut alors respecter la contrainte suivante : $Prec_i = j \leftrightarrow Succ_j = i$. Pour cela nous utilisons une contrainte de *channeling*. Cette contrainte se présente comme ceci : soient x et y deux tableaux de variables $inverseChanneling(x, y)$ assure que $x[i] = j \leftrightarrow y[j] = i$;
- L'élimination des sous-cycles au sein de la tournée est réalisée par la contrainte : $PasDeSousCycle$. Cette contrainte issue des travaux de Rousseau *et al.* (2002) [132] se décompose en deux parties, elle assure qu'un point ne peut pas être le successeur de lui-même et que si j est le successeur direct de i alors le début du chemin partiel se terminant à i et la fin du chemin partiel commençant à j ne peuvent pas être le même point.

La technique de branchement

Nous utilisons comme algorithme de recherche arborescente la technique introduite par Harvey et Ginsberg [87] appelée : *Limited Discrepancy Search* (LDS). En effet, pour attribuer à chaque variable sa valeur nous utilisons une méthode de *Branch and Bound* basée sur des techniques utilisées par la programmation par contraintes. Lorsqu'un *Branch and Bound* est effectué en programmation par contraintes, à chaque étape de la recherche une variable est instanciée et le sous-problème est alors étudié en affectant à chaque variable l'ensemble des valeurs de leur domaine. Après chaque affectation d'une valeur à une variable un processus de propagation de contraintes intervient. Ce processus prend en compte l'ensemble des contraintes et la fonction objectif du problème. La propagation aboutit à la réduction des domaines des variables et donc à la réduction de l'arbre de recherche. Lorsque, pour une variable, le domaine de définition est vidé, la solution proposée n'est pas valide, la recherche effectue donc un retour en arrière dans l'espace de recherche. La recherche arborescente peut être améliorée grâce à des heuristiques de sélection des variables et des valeurs. Les heuristiques de sélection des variables sont utilisées pour réduire l'espace de recherche et les heuristiques de sélection des valeurs sont utilisées pour guider la recherche de la solution. Ainsi une stratégie de branchement peut être d'instancier en premier lieu les variables qui sont les plus contraintes. Une autre façon de limiter l'arbre de recherche est d'utiliser des heuristiques qui évitent d'explorer des branches non prometteuses.

C'est donc pour cela que nous utilisons la technique de branchement LDS, couplée aux heuristiques de choix de variables *"MinDomain"*, *"DomOverDeg"* et *"MostConstrained"*. L'heuristique de choix : *"MinDomain"* choisit la variable ayant le plus petit domaine comme variable à instancier en premier. *"DomOverDeg"* est une heuristique qui sélectionne la variable ayant le plus petit ratio : $\frac{tailledudomaine}{degredelavariable}$, le degré d'une variable étant le nombre de contraintes dans lesquelles elle est. L'heuristique *"MostConstrained"* instancie d'abord la variable étant la plus contrainte. De plus nous utiliserons successivement les heuristiques de choix de valeurs suivantes : *IncreasingDomain* (les variables sont instanciées aux valeurs depuis la borne inférieure du domaine jusqu'à la borne supérieure du domaine), *DecreasingDomain* (les variables sont instanciées aux valeurs depuis la borne supérieure du domaine jusqu'à la borne inférieure du domaine) et *RandomIntValSelector* (les variables sont instanciées aléatoirement aux valeurs de leur domaine).

Dans la méthode LDS, une heuristique de classement de variables guide le branchement, un paramètre appelé "erreur" permet au branchement de ne pas prendre la décision induite par l'heuristique un certain nombre de fois (paramètre de la méthode pouvant varier). Nous utilisons pour notre part un paramètre d'erreur autorisée de 1, 2, ou 3. L'ensemble des combinaisons : choix de variables - choix de valeurs - nombre d'erreurs autorisées a été testé.

La liste taboue au sein de la recherche locale

Nous avons mis en place une liste taboue au sein de la recherche locale, qui permet de sortir de minima locaux lorsque le site à retirer ne peut être replacé qu'à son placement d'origine. Nous retenons alors les retraits que nous avons testés, pour ne pas avoir à refaire ces retraits si la solution courante possède toujours cette configuration. Prenons par exemple le retrait du site 20 dans la tournée 4, qui après ré-optimisation est inséré de nouveau dans la tournée 4 au même endroit dans cette tournée, car aucune autre possibilité n'est réalisable ou meilleure, on retient alors dans la liste taboue le couple (20, 4), pour qu'à l'itération suivante ce site, qui sera toujours celui qui répond au plus fort critère, ne soit pas re-testé. La liste taboue, retient tous les retraits qui ont été réalisés.

Différents paramètres de tests de la version hybride

Nous avons choisi, après différents tests sur les paramètres, d'effectuer la recherche locale toutes les 50 itérations du GRASP sur la meilleure solution obtenue par les 50 constructions. Pour une résolution du problème en juste à temps avec la méthode LNS nous itérons 70 fois le retrait de 2 sites si cela prend moins de 3 minutes de calcul, dans le cas contraire l'optimisation des 2 retraits en cours est terminé et le processus est stoppé. Pour la résolution du problème avec gestion des stocks, nous itérons 100 fois le retrait de 2 sites si cela prend moins de 3 minutes dans un premier temps. Comme les résultats obtenus ne sont pas concluants, nous avons augmenté le délai à 5 minutes. Suite à différents tests, nous avons constaté que le meilleur critère de sélection de sites pour le retrait est "plus forte contribution". Tous les résultats donnés par la suite sont ceux obtenus avec ce critère de sélection.

5.3 Les différents résultats

Dans cette partie nous allons donner les différents résultats obtenus grâce au GRASP classique puis grâce au GRASP hybride. Les méthodes testées dans ce chapitre ont pour paramètre les valeurs suivantes :

- coût de création = 2000 ;
- coût de routage = 10 ;
- coût de stockage = 10.

La valeur du paramètre α de la méthode GRASP que nous avons choisi d'utiliser est 0. Une étude plus approfondie sur les différentes valeurs que peut prendre ce paramètre de l'algorithme fera l'étude de travaux ultérieures.

Une étude sur les résultats obtenus selon les catégories d'instances et leur physionomie est faite dans le chapitre 7. Une étude sur l'impact des coûts et sur l'option de visite d'une demande par plusieurs véhicules est réalisée dans le chapitre 8.

Pour résoudre le problème par les méthodes hybrides nous avons utilisé le solveur de contraintes `CHOCO`[1]. L'ensemble des méthodes de résolution a été développé en `JAVA` et a été testé sur un Pentium IV cadencé à 2,8 GHz disposant de 512 Mo de RAM.

5.3.1 Le GRASP classique

En juste à temps

Dans le tableau 5.1, nous trouvons les résultats obtenus pour le problème en juste à temps grâce au GRASP classique. Les résultats sont séparés en fonction de la taille de la liste de candidats utilisés. Puis nous donnons pour chaque : la moyenne du coût obtenu, la moyenne du temps d'exécution, la différence en termes de coût et de temps d'exécution avec les méthodes heuristiques. Le GRASP classique permet d'obtenir en moyenne

[1] http ://choco-solver.net

Taille : 3				Taille : 5			
Moyenne coût	Moyenne temps	$\neq$ coût vs heuristiques	$\neq$ temps vs heuristiques	Moyenne coût	Moyenne temps	$\neq$ coût vs heuristiques	$\neq$ temps vs heuristiques
2 814 650	259 s	-3,5 %	$\times$ 142	2 842 621	257 s	-2,5 %	$\times$ 141

TAB. 5.1 – Résultats obtenus par le GRASP classique en juste à temps, en fonction de la taille de la liste de candidats

3,5 % de gain par rapport aux méthodes heuristiques (entre 2,9 % et 4,3 % selon les catégories d'instances). Le GRASP classique permet d'améliorer le résultat de 83 % des instances testées. Les meilleurs résultats sont obtenus lorsque l'on utilise une liste de candidats de taille 3. Une étude plus approfondie sur l'influence de ce paramètre de la méthode en fonction de la physionomie des instances est une des perspectives envisageables.

Avec gestion des stocks

Taille : 3				Taille : 5			
Moyenne coût	Moyenne temps	$\neq$ coût vs heuristiques	$\neq$ temps vs heuristiques	Moyenne coût	Moyenne temps	$\neq$ coût vs heuristiques	$\neq$ temps vs heuristiques
2 714 059	258 s	- 4,5 %	$\times$ 139	2 736 858	257 s	-3,7%	$\times$ 139

TAB. 5.2 – Résultats obtenus par le GRASP classique avec gestion des stocks, en fonction de la taille de la liste de candidats

Le tableau 5.2 permet de voir les différents résultats obtenus sur le problème avec gestion des stocks avec une résolution par GRASP classique. Les résultats sont séparés en fonction de la taille de la liste de candidats utilisés. Puis nous donnons pour chaque : la moyenne du coût obtenu, la moyenne du temps d'exécution, la différence en termes de coût et de temps d'exécution avec les méthodes heuristiques. On constate dans un premier temps que dans cette configuration de coûts, la stratégie qui consiste à livrer en avance certaines demandes, permet de baisser le coût par rapport à la stratégie de juste à temps. Ici le GRASP classique permet d'améliorer le coût des solutions d'en moyenne 4,5 % par rapport aux méthodes heuristiques (entre 3,4 % et 5 % selon les catégories d'instances). Le GRASP classique sur le problème avec gestion des stocks permet d'améliorer 90 % des instances testées avec les méthodes heuristiques.

5.3.2 Le GRASP hybride

Les tests effectués sur le GRASP hybride n'ont pas été effectués sur les 1848 fichiers présentés dans le paragraphe 3.2. En effet, les tests ont été effectués sur 363 instances parmi les 1848 instances. Nous avons sélectionné les instances n'ayant pas de fenêtres de visite trop large. Ce choix de sélection des instances a été réalisé du fait de la meilleure efficacité des techniques de programmation par contraintes sur les instances ayant des fenêtres de visite serrées. En effet, lorsque les fenêtres de visite des sites sont trop larges le fait d'avoir une variable de décision qui représente l'instant de visite de chaque site rend l'exploration trop grande. L'ensemble du domaine de la variable instant de visite est trop grand pour pouvoir obtenir une solution rapidement par énumération. Même si souvent l'ensemble des décisions prises permet de réduire le domaine des variables il reste quand même un intervalle de temps non négligeable dont chaque instant va être exploré. C'est pour cela que nos travaux vont se focaliser sur des instances n'ayant pas de fenêtres de visite larges.

En juste à temps

Dans le tableau 5.3 nous donnons les différents résultats obtenus par le GRASP hybride pour le problème en juste à temps. Les résultats sont donnés selon plusieurs critères : la taille de la liste de candidats, le temps laissé à chaque itération de LNS, le nombre d'erreurs autorisées dans LDS, l'heuristique de choix des variables lors du branchement et l'heuristique d'instanciation des valeurs des domaines des variables. Les résultats donnent l'écart

Liste Candidats taille	temps pour chaque iteration LNS	LDS paramètre	Variable choix	Valeur choix	$\neq$ Classique GRASP (coût)
5	< 3 min || 70 ×	2	MinDomain	IncreasingDomain	+5,5 %
3	< 3 min || 70 ×	3	MinDomain	IncreasingDomain	+5,5 %
3	< 3 min || 70 ×	2	MinDomain	DecreasingDomain	+5,7 %
3	< 3 min || 70 ×	2	MinDomain	RandomIntValSelector	+5,9 %
3	< 3 min || 70 ×	2	MostConstrained	IncreasingDomain	+7,7 %
3	< 3 min || 70 ×	3	MostConstrained	IncreasingDomain	+5,9 %
3	< 3 min || 70 ×	2	MostConstrained	DecreasingDomain	+5,8 %
3	< 3 min || 70 ×	1	DomOverDeg	IncreasingDomain	+6,8 %
3	< 3 min || 70 ×	2	DomOverDeg	IncreasingDomain	+7 %
3	< 3 min || 70 ×	3	DomOverDeg	IncreasingDomain	+5,1 %
3	< 3 min || 70 ×	2	DomOverDeg	DecreasingDomain	+6,8 %

TAB. 5.3 – Résultats obtenus par le GRASP hybride en juste à temps, en fonction des différents paramètres

de coût obtenu par rapport aux coûts du GRASP classique. En moyenne les résultats obtenus par le GRASP hybride sont de même qualité que ceux obtenus par les meilleures méthodes de construction et améliorations décrites dans le chapitre 1. Nous constatons ici que les meilleurs résultats pour la résolution hybride du problème en juste à temps sont obtenus par la combinaison des paramètres suivants :

- Une taille de liste de 3 candidats ;
- L'heuristique de choix des variables : DomOverDeg ;
- L'heuristique de choix des valeurs : IncreasingDomain ;
- Le choix de 3 divergences dans la technique LDS.

L'ensemble des résultats obtenus avec le GRASP hybride sur le problème en juste à temps donne en moyenne des résultats moins bons que ceux obtenus par le GRASP dit classique. Cependant nous pouvons constater sur les schémas 5.1 que les méthodes hybrides permettent de donner le meilleur résultat toutes méthodes confondues sur certaines instances.

Les schémas présents sur la figure 5.1 représentent la répartition des meilleures solutions trouvées selon leur méthode de résolution. Le premier schéma détaille l'ensemble des méthodes utilisées tandis que le second généralise selon les types (méthode de construction et améliorations "C & A", GRASP classique et GRASP hybride). Pour comprendre le premier schéma il est nécessaire de donner les explications suivantes : chaque nom de méthode commence par son type : C & A pour construction et améliorations, "Class" pour GRASP classique et "Hybr" pour GRASP hybride. L'intitulé "Class" et "Hyb" sont suivies de la taille de la liste de candidats utilisée. Puis pour les méthodes hybrides sont donnés dans l'ordre : l'heuristique de choix de variables (MD pour MinDomain, DOD pour DomOverDeg et MC pour MostConstrained), l'heuristique de choix de valeurs (ID pour IncreasingDomain, DD pour DecreasingDomain et R pour RandomIntValSelector) et le nombre de divergences maximum autorisées pour LDS.

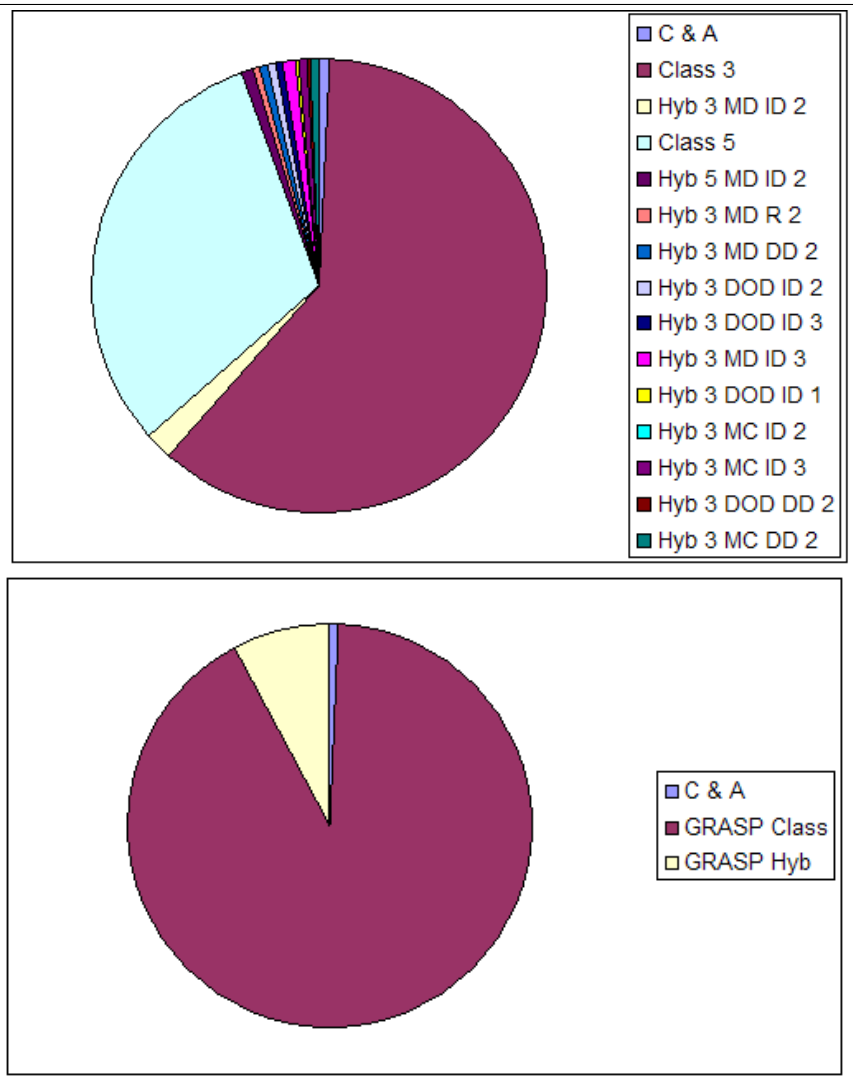
Dans ce chapitre nous raisonnons globalement, plus de détails concernant les résultats selon les instances sont donnés dans le chapitre 7.

Avec gestion des stocks

Dans le tableau 5.4 nous donnons les résultats obtenus avec le GRASP hybride sur le problème avec gestion des stocks. Les résultats sont donnés selon plusieurs critères : la taille de la liste de candidats, le temps laissé à chaque itération de LNS, le nombre d'erreurs autorisées dans LDS, l'heuristique de choix des variables lors du branchement, l'heuristique de choix des valeurs et l'écart de coût obtenu par rapport aux coûts du GRASP classique. Les résultats obtenus ici restent éloignés de ceux obtenus par la méthode classique. Nous constatons néanmoins que la combinaison qui donne les meilleurs résultats est : DomOverDeg comme heuristique de choix de variable, IncreasingDomain comme heuristique de choix des valeurs et une divergence autorisée dans LDS.

Dans le schéma 5.2 nous trouvons la répartition des meilleurs résultats obtenus selon les différentes méthodes.

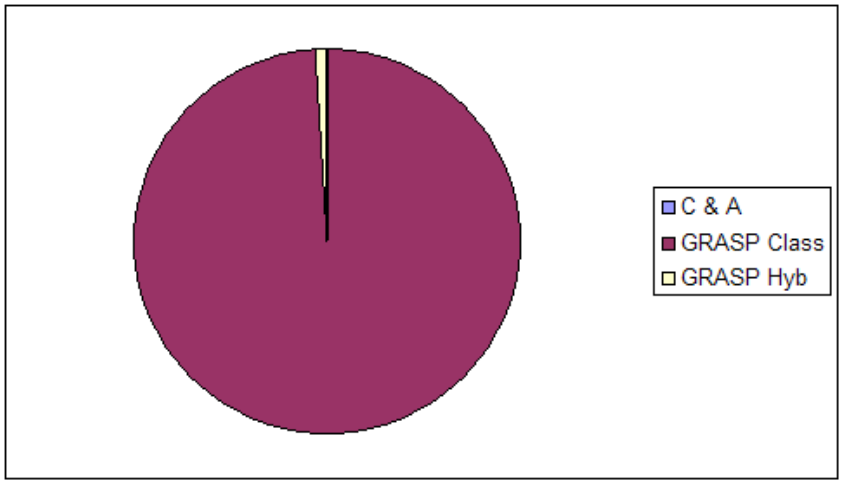
Figure 5.1 *Comparaison du nombre de solutions minimum trouvé pour chaque méthode de résolution : GRASP en juste à temps*



Liste Candidats taille	temps pour chaque itération LNS	LDS paramètre	Variable choix	Valeur choix	≠ Classique GRASP (coût)
3	< 3 min || 100 ×	3	MinDomain	IncreasingDomain	+11,7%
3	< 5 min|| 100 ×	3	MinDomain	IncreasingDomain	+10,7 %
3	< 5 min || 100 ×	5	MinDomain	IncreasingDomain	+10,5 %
3	< 5 min || 100 ×	1	MinDomain	IncreasingDomain	+12,2 %
3	< 5 min || 100 ×	2	DomOverDeg	IncreasingDomain	+ 13 %
3	< 5 min || 100 ×	1	DomOverDeg	IncreasingDomain	+ 10,2 %
3	< 5 min || 100 ×	3	DomOverDeg	IncreasingDomain	+ 10,8 %
3	< 5 min || 100 ×	1	DomOverDeg	DecreasingDomain	+ 10,6 %
3	< 5 min || 100 ×	1	MostConstrained	IncreasingDomain	+ 10,9 %
3	< 5 min || 100 ×	2	MostConstrained	IncreasingDomain	+ 11,7 %

TAB. 5.4 – Résultats obtenus par le GRASP hybride avec gestion des stocks, en fonction des différents paramètres

Figure 5.2 *Comparaison du nombre de solutions minimum trouvé pour chaque méthode de résolution : GRASP avec gestion des stocks*



Nous constatons que la version hybride donne moins de fois la meilleure solution parmi toutes pour le problème dans la version avec gestion des stocks contrairement à la la méthode en juste à temps.

5.4 Conclusions et perspectives

Dans ce chapitre, nous avons expliqué comment nous avons résolu le problème de construction de tournées dans le cadre de la logistique inverse grâce à une métaheuristique nommée GRASP. Cette métaheuristique se décompose en deux parties : construction de solution et recherche locale. La construction de solution a toujours été réalisée par la technique de *meilleure insertion*. La recherche locale a été traitée de deux manières différentes : tout d'abord grâce à des techniques d'amélioration classiques issues de la recherche opérationnelle, puis grâce à des techniques de programmation par contraintes. Les résultats obtenus avec des techniques classiques sont concluants. Les résultats obtenus grâce à la méthode hybride sont prometteurs, en effet nous avons pu constater qu'elles apportent dans plusieurs cas la meilleure solution parmi toutes les méthodes testées. Les méthodes hybrides s'avèrent quand même plus efficace dans la version en juste à temps du problème que dans la version avec gestion des stocks.

Une première perspective de travail sur la méthode de résolution hybride serait de développer une contrainte globale permettant de gérer d'un seul bloc le problème de ré-optimisation lors de la recherche locale. Grâce à cette contrainte globale plus de filtrage pourrait être réalisé.

Une seconde perspective serait d'utiliser la contrainte globale de chemin développée par Beldiceanu *et al.* [11]. Dans leurs travaux les auteurs s'attachent à montrer comment prendre en compte de manière globale un certains nombre de restrictions (précédence entre sommets, incomparabilités entre sommets) au sein d'une contrainte de partitionnement de graphes par les arbres.

Chapitre 6

Résolution basée sur les méthodes exactes

Le programme linéaire (*cf.* 1.3.3) correspondant à notre problème est de trop grande taille pour être résolu directement par l'algorithme du simplexe. C'est pourquoi, nous avons décidé d'utiliser une méthode de décomposition pour le résoudre. Nous avons donc utilisé la technique de génération de colonnes (induite par la décomposition de Dantzig-Wolfe (1960) [41]). À la fin de la résolution du problème relaxé par génération de colonnes nous résolvons le problème non relaxé par *Branch and Bound* [102].

Dans ce chapitre nous allons dans une première partie rappeler les principes de cette technique. Puis nous verrons son adaptation à notre problème dans ses deux versions : en juste à temps et avec gestion des stocks. La résolution du sous-problème s'effectuera en utilisant la programmation dynamique, la recherche taboue et des techniques de programmation par contraintes.

6.1 La technique de génération de colonnes

La génération de colonnes a été utilisée pour la première fois par Gilmore et Gomory (1961, 1963 [71], [72]). Minoux (1983) [113] et Barnhart *et al.* (1998) [7] en font une bonne description générale. Plus récemment le livre de Desaulniers *et al.* (2005) [46] est consacré totalement à cette méthode.

La génération de colonnes est une méthode pour résoudre efficacement les programmes linéaires de grande taille. Elle repose sur la décomposition de Dantzig-Wolfe (1960) [41]. Cette méthode s'applique à des problèmes de partitionnement dont la formulation implique un nombre important de variables. En effet, ce grand nombre de variables empêche la génération explicite de toutes les colonnes (Ω) de la matrice des contraintes : A. A est donc connue implicitement, mais on ne connaîtra qu'un sous-ensemble $\Omega_1 \subseteq \Omega$ de colonnes, dont on prouvera qu'elles permettent d'obtenir la solution optimale du problème.

On débute la résolution du programme linéaire continu (appelé Programme Maître : PM) en partant d'une solution réalisable (sous-ensemble de la matrice A), dans le but de pouvoir caractériser d'autres solutions améliorantes. La solution réalisable utilisée à l'initialisation de l'algorithme est souvent obtenue grâce à une heuristique. Le PM associé au sous-ensemble réalisable de la matrice A est appelé : le Programme Maître Restreint (PMR). Il est résolu par l'algorithme du simplexe. En connaissant un ensemble de colonnes Ω_1 donné, un générateur de colonnes est alors nécessaire pour proposer une ou plusieurs colonnes qui amélioreront la solution courante. Ce générateur de colonnes est appelé le "sous-problème" associé à la formulation de départ. Les nouvelles colonnes améliorantes sont donc générées par le sous-problème du programme maître. Le Problème Maître ne prend en compte aucune contrainte pour s'assurer de la faisabilité des colonnes. C'est le sous-problème qui doit en tenir compte et qui doit vérifier qu'il génère des colonnes réalisables. Ces colonnes sont calculées à partir des variables duales de la solution courante et sont ensuite ajoutées au Programme Maître Restreint, qui est ensuite ré-optimisé grâce au simplexe. Ce processus est itéré tant qu'il existe des colonnes améliorantes.

Les performances de ce type de résolution dépendent pour beaucoup de l'algorithme utilisé pour la résolution du sous-problème. À la fin du processus, une solution optimale est obtenue pour le Programme Maître Restreint. Le processus de génération de colonnes permet d'obtenir la solution optimale du problème relaxé, cette solution est donc une borne inférieure du problème entier. Il est donc nécessaire, ensuite, d'utiliser la technique du *Branch And Price*, pour obtenir la solution optimale entière du problème initial.

Nous allons prendre ici un exemple de problème de construction de tournées simples (appelons P ce problème) et ceci avec une flotte illimitée. Une seule contrainte est ici à vérifier : chaque client doit être visité une fois. L'objectif est de trouver une solution minimisant le coût des tournées sélectionnées.
Posons alors :

- Ω : ensemble des tournées réalisables ;
- N : ensemble des clients à servir ;
- c_r : coût de la tournée r ;
- x_r : variable binaire égale à 1 si la tournée r est sélectionnée dans la solution, 0 sinon ;
- a_{ir} : constante binaire valant 1 si le client i est visité par la tournée r, 0 sinon.

$$\text{Minimiser : } z = \sum_{r \in \Omega} c_r x_r \tag{6.1}$$

Sous Contraintes :

$$\forall\, i \in N \quad \sum_{r \in \Omega} x_r a_{ir} = 1 \tag{6.2}$$

$$\forall\, r \quad x_r \in \{0, 1\} \tag{6.3}$$

On retrouve ici en 6.2, la contrainte permettant que chaque client soit visité une seule fois. Le but de ce problème est de minimiser le coût total des tournées sélectionnées (6.1).

Dans une version notée $P^{'}$ du problème P on remplacera la contrainte de partitionnement 6.2 par la contrainte de recouvrement suivante :

$$\forall\, i \in N \quad \sum_{r \in \Omega} x_r a_{ir} \geq 1 \tag{6.4}$$

Nous relaxerons également les contraintes d'intégrité de x_r. Ainsi nous pouvons écrire $P^{'}$ comme ceci :

$$\text{Minimiser : } z = \sum_{r \in \Omega} c_r x_r \tag{6.5}$$

Sous Contraintes :

$$\forall\, i \in N \quad \sum_{r \in \Omega} x_r a_{ir} \geq 1 \tag{6.6}$$

$$\forall\, r \quad x_r \geq 0 \tag{6.7}$$

L'énumération complète des tournées de Ω est impossible dans un temps raisonnable. Le principe de la génération de colonnes est de ne considérer qu'un sous-ensemble de colonnes qui sont prometteuses et à chaque itération il s'agit de faire entrer une nouvelle variable en base. Chaque itération de la génération de colonnes consiste à optimiser le programme maître restreint de façon à avoir la solution optimale courante et les variables duales associées et de trouver une colonne dont le coût réduit est négatif. C'est le générateur de colonnes, qui va se charger de fournir les bonnes tournées nécessaires à la résolution par la méthode de génération de colonnes. Ainsi dans le petit exemple donné au dessus le sous-problème génère des tournées respectant la capacité du véhicule et dont l'origine et l'arrivée sont le dépôt.

Écrivons maintenant le dual noté D du problème P'. Soit μ_i la variable duale associée à la contrainte 6.6. Nous pouvons alors écrire D ainsi :

$$\text{Maximiser : } z = \sum_{i \in N} \mu_i \tag{6.8}$$

Sous Contraintes :

$$\forall\, r \in \Omega \quad \sum_{i \in N} \mu_i a_{ir} \leq c_r \tag{6.9}$$

$$\forall\, i \quad \mu_i \geq 0 \tag{6.10}$$

(6.11)

Trouver une colonne qui améliore le coût de $P^{'}$ revient à trouver une colonne r telle que :

$$c_r - \sum_{i \in N} \mu_i a_{ir} \leq 0 \tag{6.12}$$

Pour trouver une colonne améliorante au problème $P^{'}$ nous devons trouver la colonne de coût $c_r - \sum_{i \in N} \mu_i a_{ir}$ minimal. On se rend compte ici que ce coût peut se décomposer aisément en discrétisant les coûts induits par les choix de successeur dans la tournée. En effet si on note x_{ijr} la variable binaire qui est égale à 1 si j est immédiatement visité après i dans la tournée r et égale à 0 sinon, on a alors $\sum_{i \in N} \sum_{j \in N} \bar{c_{ij}} x_{ijr} = c_r - \sum_{i \in N} \mu_i a_{ir}$ où $\bar{c_{ij}} = c_{ij} - \mu_i a_{ir}$. On s'aperçoit alors que trouver une colonne améliorante pour $P^{'}$ revient à résoudre le problème du plus court chemin élémentaire (car les coûts sur les arcs peuvent être négatifs) sous contrainte de ressources dans un graphe où chaque arc (i, j) est pondéré par le coût : $\bar{c_{ij}}$.

Dans la suite de ce chapitre, nous allons montrer comment nous avons utilisé la technique de génération de colonnes pour résoudre tout d'abord le problème étudié en juste à temps puis avec gestion des stocks. Nous aborderons ensuite le problème de la dégénérescence qui peut survenir et enfin nous verrons comment nous utilisons la technique du *Branch and Bound* [102] pour trouver une solution entière au problème. En effet nous n'envisageons pas dans le cadre de cette thèse de développer une procédure complète de *Branch and Price* (Barnhart *et al.*, 1998 [7] et Vanderbeck, 2000 [159]).

6.2 La résolution du problème en juste à temps

Considérons dans un premier temps le problème dans sa version en juste à temps. Nous allons voir sa modélisation puis nous décrirons le sous-problème qui lui est associé et enfin nous verrons les différentes techniques que nous avons utilisées pour résoudre le sous-problème : méthode taboue, programmation dynamique et programmation par contraintes.

6.2.1 Le problème et sa modélisation

Dans cette première modélisation nous considérons que les sites ont une demande fixe en produit pour chaque jour qui est connue et que pour chaque jour de la planification les sites sont livrés de leur besoin (*c.-à-d.* pas possibilité de stockage préalable ni de coût de stockage dans ce cas là). Par simplification, nous imposons que les quantités livrées ou collectées soient des valeurs entières. Nous résolvons ainsi, le problème sans anticipation des demandes. Les demandes restent néanmoins préemptives. Les quantités collectées ne sont pas imposées elles sont induites par la capacité de stockage du site.

Dans cette modélisation une tournée est représentée, pour chaque site, par la quantité de produits qui est livrée et collectée (si une tournée ne passe pas par un site sa quantité livrée ou collectée est donc nulle) et les instants de visite de chaque site. Ainsi, chaque tournée devra respecter les contraintes de capacité des véhicules, des fenêtres de visite des sites. Notons :

- I : ensemble des sites ;
- V : nombre de véhicules disponibles dans la flotte ;
- T : horizon de la planification ;
- Ω : ensemble des tournées réalisables ;
- x_{rt} : variable binaire égale à 1 si la tournée r est sélectionnée pour être réalisée le jour t, 0 sinon (unique variable du problème) ;
- a^1_{ir} : quantité de produits *livrée* au client i par la tournée r ;
- a^2_{ir} : quantité de matériaux (produits en retour et palettes vides) *collectée* au client i par la tournée r ;
- c_r : coût de la tournée r comprenant le coût fixe d'utilisation du véhicule et le coût de routage (proportionnel à la distance parcourue) ;
- d_{it} : demande de livraison du client i le jour t ;
- $Capa_i$: capacité de stockage du site i ;
- Pdt_{0i} : état du stock de produit du client i au début de la planification (les demandes sont mises à jour en fonction de cette valeur) ;

- Mat_{0i} : état du stock de matériaux (palettes vides et produits en retour) du client i au début de la planification ;
- Mat_{ki} : le nombre de matériaux (palettes vides et produits en retour) qui s'ajoute chez le client i le jour k ;
- $Stock_{0i}$: état du stock total du site i au début de la planification
  $Stock_{0i} = Pdt_{0i} + Mat_{0i}$.

Le modèle de partitionnement correspondant à notre problème en juste à temps dans les livraisons peut donc s'écrire comme suit :

$$\text{Min :} \qquad z = \sum_{r\in\Omega}\sum_{t\in[1,T]} c_r x_{rt} \tag{6.13}$$

Sous Contraintes :

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega} x_{rt} a^1_{ir} \geq d_{it} \tag{6.14}$$

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} x_{rk} a^2_{ir} \leq \sum_{k=1}^{t} Mat_{ki} \tag{6.15}$$

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} (a^1_{ir} - a^2_{ir}) x_{rk} \leq Capa_i + \sum_{k=1}^{t} d_{ik} - Stock_{0i} \tag{6.16}$$

$$\forall\, t \in [1,T] \qquad \sum_{r\in\Omega} x_{rt} \leq V \tag{6.17}$$

$$x_{rt} \in \aleph \tag{6.18}$$

Pour faciliter l'écriture de notre modèle nous allons introduire les trois termes suivants :

- $S'_{it} = Capa_i + \sum_{k=1}^{t} d_{ik} - Stock_{0i}$
- $A_{ir} = a^1_{ir} - a^2_{ir}$
- $M_{ti} = \sum_{k=1}^{t} Mat_{ki}$

Ainsi, nous pouvons réécrire le modèle comme suit :

$$\text{Min :} \qquad z = \sum_{r\in\Omega}\sum_{t\in[1,T]} c_r x_{rt} \tag{6.19}$$

Sous Contraintes :

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega} x_{rt} a^1_{ir} \geq d_{it} \tag{6.20}$$

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} x_{rk} a^2_{ir} \leq M_{ti} \tag{6.21}$$

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} A_{ir} x_{rk} \leq S'_{it} \tag{6.22}$$

$$\forall\, t \in [1,T] \qquad \sum_{r\in\Omega} x_{rt} \leq V \tag{6.23}$$

$$x_{rt} \in \aleph \tag{6.24}$$

L'objectif est de minimiser les coûts de routage sous les contraintes suivantes :

- Contrainte 6.20 : Respect de la demande de chaque site pour chaque jour ;
- Contrainte 6.21 : Respect de la quantité possible à collecter de chaque site pour chaque jour ;
- Contrainte 6.22 : Respect de la capacité de stockage de chaque site ;

– Contrainte 6.23 : Respect de la taille de la flotte de véhicules.

Nous vérifions pour chaque jour que la capacité est respectée mais au sein d'une journée nous n'entrons pas dans les détails. Si deux tournées passent par le même site nous vérifions que globalement la capacité est vérifiée mais entre le passage de la première tournée et la seconde il se peut que la capacité ne soit pas respectée.

Associons à la contrainte 6.20 la variable duale μ_{it}, à la contrainte 6.21 la variable duale σ_{it}, à la contrainte 6.22 : β_{it} et à la contrainte 6.23 la variable duale α_t.

Nous résumons dans le tableau 6.1 le schéma général de la matrice des contraintes correspondant au problème maître.

Nous pouvons alors écrire le dual du problème ainsi :

$$\text{Max :} \qquad z' = \sum_{i\in I}\sum_{t\in[1,T]} d_{it}\mu_{it} - \sum_{i\in I}\sum_{t\in[1,T]} M_{ti}\sigma_{it} - \sum_{i\in I}\sum_{t\in[1,T]} S'_{it}\beta_{it} - \sum_{t\in[1,T]} V\alpha_t \quad (6.25)$$

Sous Contraintes :

$$\forall\, r \in \Omega \quad \forall\, t \in [1,T] \quad \sum_{i\in I} a^1_{ir}\mu_{it} - \sum_{i\in I}\sum_{k=t}^{T} a^2_{ir}\sigma_{ik} - \sum_{i\in I}\sum_{k=t}^{T} A_{ir}\beta_{ik} - \alpha_t \le c_r \quad (6.26)$$

$$\forall\, i,t \qquad \mu_{it} \ge 0 \quad (6.27)$$

$$\forall\, i,t \qquad \sigma_{it} \ge 0 \quad (6.28)$$

$$\forall\, i,t \qquad \beta_{it} \ge 0 \quad (6.29)$$

$$\forall\, t \qquad \alpha_t \ge 0 \quad (6.30)$$

L'objectif du sous-problème est donc de trouver des tournées de coût réduit négatif, c'est à dire des tournées r telles que :

$$c_r - (\sum_{i\in I} a^1_{ir}\mu^*_{it} - \sum_{i\in I}\sum_{k=t}^{T} a^2_{ir}\sigma^*_{ik} - \sum_{i\in I}\sum_{k=t}^{T} A_{ir}\beta^*_{ik} - \alpha^*_t) \le 0 \quad (6.31)$$

Ceci est équivalent à :

$$c_r - (\sum_{i\in I}(a^1_{ir}\mu^*_{it} - \sum_{k=t}^{T} a^2_{ir}\sigma^*_{ik} - \sum_{k=t}^{T} A_{ir}\beta^*_{ik}) - \alpha^*_t) \le 0 \quad (6.32)$$

Notons CF le coût fixe de création d'une tournée, x^r_{ij} est une variable binaire égale à 1 si la tournée r visite j immédiatement après i et c_{ij} le coût associé au parcours de de la distance séparant i et j. On peut alors exprimer le coût réel d'une tournée ainsi :

$$\forall\, r \in \Omega \quad \sum_{i\in N}\sum_{j\in N} x^r_{ij}c_{ij} + CF = c_r$$

6.2.2 La description du sous-problème

Tout comme nous l'avons vu dans la partie 6.1 de ce chapitre, trouver une colonne améliorante peut pouvoir se ramener à rechercher un plus court chemin élémentaire sous contraintes de ressources. Nous allons dans cette partie montrer quel est le graphe et les coûts associés à ce graphe qui permettent de trouver une colonne améliorante pour notre problème.

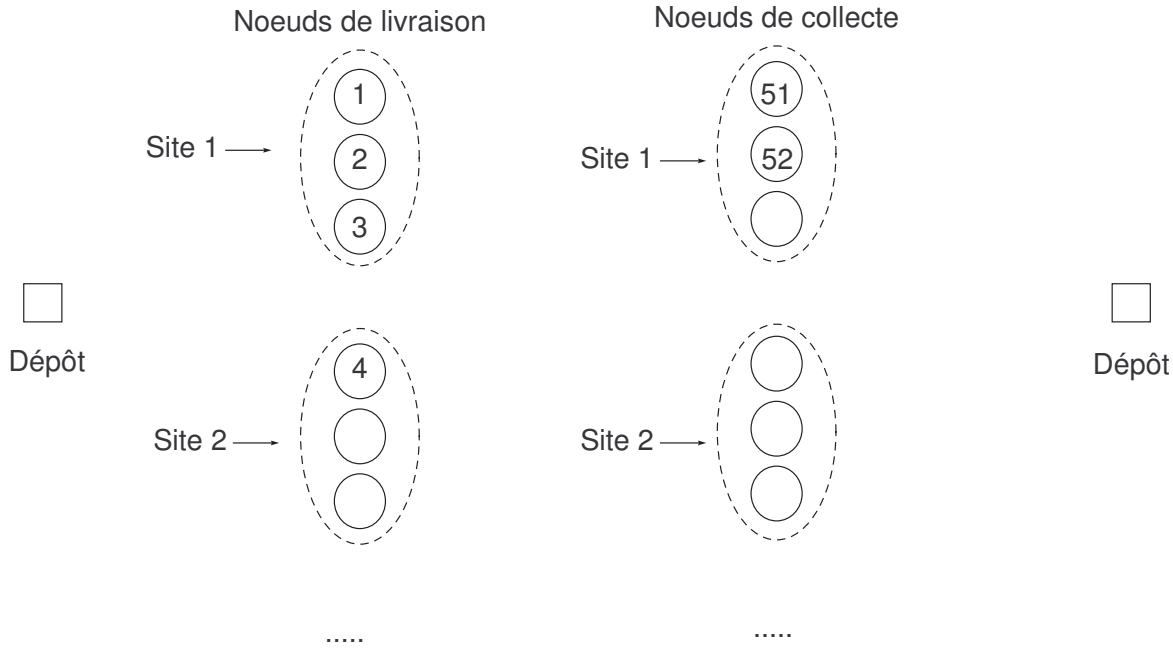
Soit $G = (V', A)$ le graphe où V' représente l'ensemble des nœuds de notre problème et A l'ensemble des arcs réalisables. Deux actions sont possibles dans chaque site : livraison de produits neufs et collecte de matériaux (palettes vides et retours de produits). C'est pourquoi un site est divisé en deux nœuds pour séparer sa

TAB. 6.1 – Tableau de la matrice des contraintes du problème maître du problème en juste à temps

			Tournées																				
			jour 1					**jour 2**					**…**	**jour T**							**type**	**second**	
			1	**…**	**r**	**…**	**w**	**1**	**…**	**r**	**…**	**w**		**1**	**…**	**r**	**…**	**w**			**contrainte**	**membre**	**Duales**
	i=1	t=1	a^1_{11}	…	a^1_{1r}	…	a^1_{1w}						…								≥	d_{11}	μ_{11}
I * T		t=2						a^1_{11}	…	a^1_{1r}	…	a^1_{1w}									≥	d_{12}	μ_{12}
contraintes		…																			…	…	…
respect		t=T												a^1_{11}	…	a^1_{1r}	…	a^1_{1w}			≥	d_{1T}	μ_{1T}
demande	i=2	t=1	a^1_{21}	…	a^1_{2r}	…	a^1_{2w}														≥	d_{21}	μ_{21}
	…	…	…	…	…	…	…	…	…	…	…	…									…	…	…
	i =1	t=1	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}														≤	M_{11}	σ_{11}
I * T		t=2	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}									≤	M_{12}	σ_{12}
contraintes		…											…								…	…	…
respect		t=T	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}		a^2_{11}	…	a^2_{1r}	…	a^2_{1w}			≤	M_{1T}	σ_{1T}
quantité	i=2	t=1	a^2_{21}	…	a^2_{2r}	…	a^2_{2w}														≤	M_{21}	σ_{21}
collectée	…	…																			…	…	…
	i =1	t=1	A_{11}	…	A_{1r}	…	A_{1w}														≤	S'_{11}	β_{11}
I * T		t=2	A_{11}	…	A_{1r}	…	A_{1w}	A_{11}	…	A_{1r}	…	A_{1w}									≤	S'_{12}	β_{12}
contraintes		…											…								…	…	…
respect		t=T	A_{11}	…	A_{1r}	…	A_{1w}	A_{11}	…	A_{1r}	…	A_{1w}		A_{11}	…	A_{1r}	…	A_{1w}			≤	S'_{1T}	β_{1T}
capacité	i=2	t=1	A_{21}	…	A_{2r}	…	A_{2w}														≤	S'_{21}	β_{21}
stockage	…	…																			…	…	…
T		t=1	1	…	1	…	1														≤	V	α_1
contraintes		t=2						1	…	1	…	1									≤	V	α_2
respect		…											…								…	…	…
flotte		t=T												1	…	1	…	1			≤	V	α_T

partie collecte de sa partie livraison, ainsi une tournée peut passer plusieurs fois par un même site réel (une fois pour livrer et une autre fois pour collecter) mais une seule fois par site pour chaque action. De plus, les quantités livrées et collectées ne sont pas déterminées par avance. Notons P_{it} l'ensemble des valeurs possibles pour la quantité de livraison d'une tournée au site i le jour t et C_{it} l'ensemble des valeurs possibles pour la quantité collectée d'une tournée au site i le jour t. Pour chaque site de livraison nous avons donc autant de nœuds de livraison que de quantité possible. Il en est de même pour les sites de collectes et leur déclinaison en nœuds de collectes selon les quantités. Ainsi dans notre graphe nous avons $2 + \sum_{i \in I, t \in T} (C_{it} + P_{it})$ nœuds. Les nœuds i et j correspondent à un site auquel on lui associe grâce à son numéro : une action (collecte ou livraison) et une quantité. c_{ij} est le coût associé à cet arc. Nous pouvons voir sur la figure 6.1 la physionomie de notre réseau sans les arcs pour le moment.

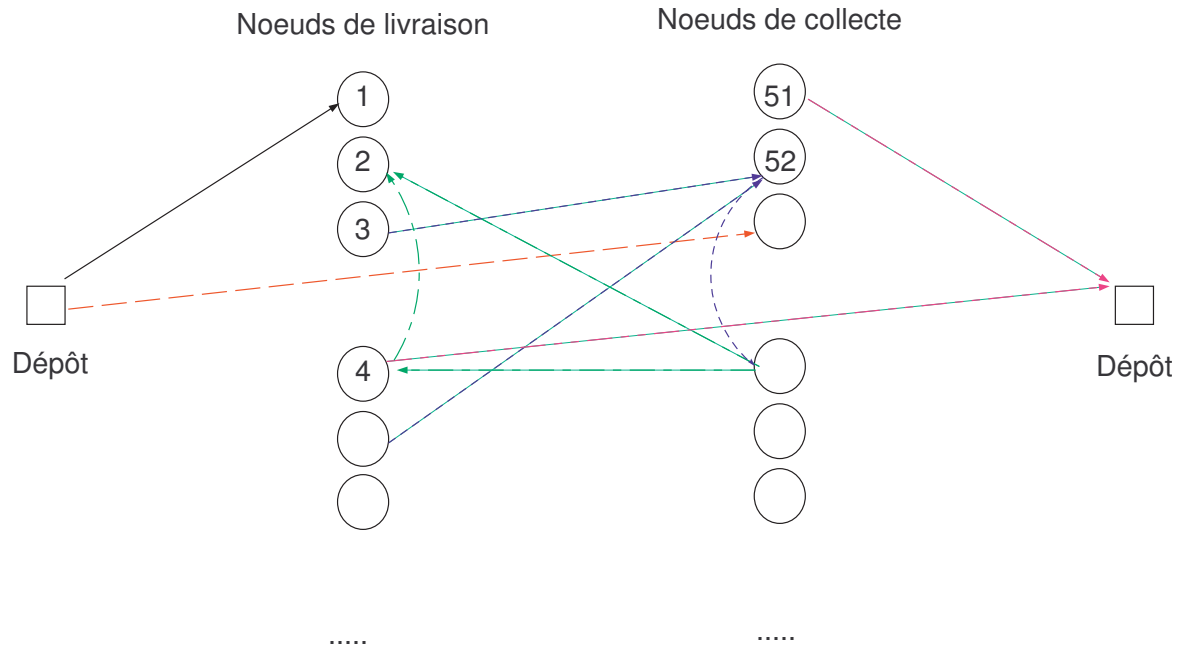
Figure 6.1 *Représentation des nœuds du graphe pour chaque jour*



Notons, a_j^1 la quantité livrée au nœud j et a_j^2 la quantité collectée au site j. Pour chaque jour t nous pouvons créer un graphe G^t où les différents coûts vont être répartis en 5 types comme suit (les coûts sont portés par l'arc entrant des sites) :

- Pour tous les arcs (i, j) où i est le dépôt de départ et j est un nœud de livraison le coût associé est : $c_{ij} + CF - a_j^1 \mu_{jt}^* + \sum_{k=t}^{T} a_j^1 \beta_{jk}^*$;
- Pour tous les arcs (i, j), où i est le dépôt de départ et j un nœud de collecte le coût associé est de : $c_{ij} + CF + \sum_{k=t}^{T} a_j^2 \sigma_{jk}^* - \sum_{k=t}^{T} a_j^2 \beta_{jk}^*$;
- Pour tous les arcs (i, j) avec i un nœud autre que le dépôt de départ ou d'arrivée et j un nœud de livraison le coût est : $c_{ij} - a_j^1 \mu_{jt}^* + \sum_{k=t}^{T} a_j^1 \beta_{jk}^*$;
- Pour tous les arcs (i, j) avec i un nœud autre que le dépôt de départ ou d'arrivée et j un nœud de collecte le coût est : $c_{ij} + \sum_{k=t}^{T} a_j^2 \sigma_{jk}^* - \sum_{k=t}^{T} a_j^2 \beta_{jk}^*$;
- Pour tous les arcs (i, j) avec i un nœud autre que le dépôt de départ ou d'arrivée et j le dépôt d'arrivée le coût est : $c_{ij} + \alpha_t^*$.

Figure 6.2 *Représentation simplifiée du graphe pour chaque jour : un seul arc par type est représenté ici.*



La figure 6.2 donne un exemple de graphe où seulement un arc par type est représenté. Nous trouvons les types suivants :

- arc allant du dépôt de départ vers un nœud de livraison ;
- arc allant du dépôt de départ vers nœud de collecte ;
- arc entrant dans un nœud de livraison (dont l'origine n'est pas le dépôt de départ) ;
- arc entrant dans un nœud de collecte (dont l'origine n'est pas le dépôt de départ) ;
- arc entrant dans le dépôt d'arrivée.

Les arcs entre nœud de livraison d'un même site n'existent pas et il en est de même pour ceux entre les nœuds de collecte d'un même site, ceci dans le but d'interdire de passer deux fois par le même site au sein d'une même tournée. De plus, comme il est dans notre problème identique de faire une collecte suivie immédiatement d'une livraison dans un même site ou une livraison puis immédiatement une collecte dans un même site, nous choisissons de garder qu'une seule des deux possibilités dans notre graphe, celle qui nous paraît le plus logique c'est à dire l'enchaînement livraison puis collecte.

Le but du sous-problème est de générer la meilleure tournée réalisable au sein de ce graphe. Les colonnes générées doivent avoir un coût réduit négatif. Les nouvelles colonnes (tournées) proposées devront respecter les contraintes de fenêtre horaire des sites visités et de capacité du véhicule. À chaque itération la méthode du Simplexe détermine les valeurs des variables duales définies ci-dessus pour les contraintes 6.20, 6.21, 6.22, 6.23. Soient :

- x_{ij} variable égale à 1 si l'arc (i, j) est parcouru par le véhicule dans la tournée ;
- t_i l'instant de visite du site i dans la nouvelle tournée générée ;
- a_i et b_i respectivement la borne inférieure et supérieure de la fenêtre de visite du site i ;
- a_j^1 la quantité livrée à j ;
- a_j^2 la quantité collectée chez j ;
- c_{ij} le coût associé au parcours de l'arc (i, j),
- CL_i la capacité libre dans le véhicule à son départ du site i ;
- CapaVeh la capacité des véhicules ;
- s_i le temps de service du client i ;
- t_{ij} le temps de parcours de la distance séparant i et j ;
- M une constante très grande ($M >> 0$) ;
- I ensemble des sites (0 étant le dépôt).

On peut alors formuler le sous-problème, caractérisant une tournée pour la journée t, ainsi :

$$\text{Min :} \quad \sum_{i\in A}\sum_{j\in A} c_{ij}x_{ij} - \sum_{i\in A}\sum_{j\in A}(a_j^1\mu_{jt}^* - \sum_{k=t}^{T} a_j^2\sigma_{jk}^* - \sum_{k=1}^{T}(a_j^1 - a_j^2)\beta_{jk}^*)x_{ij} - \alpha_t^* \tag{6.33}$$

Sous Contraintes :

$$\forall\, j \in A \quad \sum_{i\in A} x_{ij} - \sum_{i\in A} x_{ji} = 0 \tag{6.34}$$

$$\sum_{i\in A} x_{0i} + \sum_{i\in A} x_{i0} = 2 \tag{6.35}$$

$$\forall\, i \in A,\ j \in A \quad CL_j x_{ij} = (CL_i - a_j^1 + a_j^2)x_{ij} \tag{6.36}$$

$$\forall\, i \in A \quad 0 \leq CL_i \leq CapaVeh \tag{6.37}$$

$$CL_0 = \sum_{i\in A} a_i^1 \tag{6.38}$$

$$\forall\, i \in A,\ j \in A \quad t_i + s_i + t_{ij} - (1 - x_{ij})M \leq t_j \tag{6.39}$$

$$\forall\, i \in A \quad a_i \leq t_i \leq b_i \tag{6.40}$$

$$\forall\, i \in A,\ j \in A \quad x_{ij} \in \{0, 1\} \tag{6.41}$$

L'objectif est ici de trouver des tournées dont le coût réduit est négatif (6.33). La tournée générée doit respecter la conservation des flux (6.34). La tournée doit débuter et terminer au dépôt (6.35). La capacité du véhicule doit être respectée (6.36, 6.37, 6.38). Les tournées doivent respecter les fenêtres de visites des sites (6.39, 6.40).

6.2.3 Implémentation

Le graphe du sous-problème peut très vite prendre des proportions énormes. C'est pourquoi dans un souci de simplification de celui-ci, nous avons restreint les quantités possibles de livraison à 0, 1/2 de la demande ou à la demande tout entière et les quantités de collectes à rien ou la demande entière. Ainsi pour chaque site du problème il y aura 2 nœuds correspondant aux nœuds de livraison et 1 nœud correspondant au nœud de collecte. Ce qui fait que pour une instance à 5 sites par exemple nous avons 17 nœuds dans le graphe (5×3 pour les sites et 2 pour le dépôt de départ et d'arrivée). Cette hypothèse ne nous semble pas trop restrictive en pratique.

Dans la suite de ce chapitre nous allons étudier plusieurs manières de résoudre le sous-problème. Tout d'abord nous allons utiliser la programmation dynamique. Puis nous allons résoudre le sous-problème grâce à la méthode de recherche taboue. Et enfin nous verrons comment employer des techniques de programmation par contraintes sur ce problème. Dans toutes les descriptions des méthodes données dans la suite de ce chapitre nous utilisons les notations introduites dans la partie 6.2.2.

6.2.4 La résolution du sous-problème par programmation dynamique

La première méthode utilisée pour résoudre le problème de plus court chemin dans le graphe décrit dans la partie 6.2.2 est la programmation dynamique. Il existe différents algorithmes de programmation dynamique adapté à notre problème (Desrochers [47], [48] et Feillet [60]). L'algorithme de Desrochers ([47], [48]) est basé sur celui proposé par Bellman. L'objectif du problème est de construire un plus court chemin entre une origine o et une destination d de coût minimal, satisfaisant toutes les contraintes de ressources. L'algorithme de Desrochers [47] permet d'obtenir tous les chemins optimaux joignant le sommet de départ au sommet d'arrivée. Pour cela, l'algorithme fonctionne sur le principe de correction d'étiquettes. Pour chaque sommet du graphe l'algorithme associe un ensemble de labels correspondant à un ensemble de chemins optimaux (non dominés) permettant d'arriver à ce sommet. On dit qu'un label est non dominé lorsqu'il n'existe pas d'autre chemin consommant moins ou autant de chacune des ressources. Les étiquettes associées au site d'arrivée correspondent donc aux chemins optimaux.

Reprenons ici la notation simplifiée proposée par Desrochers [47]. À chaque chemin X_{oj} d'origine o et d'extrémité j est associé un label $(T_j^1, T_j^2, \ldots, T_j^L, C_j)$ où $T_j^1, T_j^2, \ldots, T_j^L$ représentent l'état de consommation des L ressources dans le chemin considéré (*c.-à-d.* les quantités consommées de chacune des ressources disponibles) et C_j est égal au coût du chemin. X_{oj} est caractérisé par l'état $R_j = (T_j^1, T_j^2, \ldots, T_j^L)$. Considérons maintenant deux chemins distincts $X_{oj}^{'}$ et X_{oj} allant de o jusqu'à j et leur label associé $(R_j^{'}, C_j^{'})$ et (R_j, C_j). $X_{oj}^{'}$ domine X_{oj} (s'écrit aussi $X_{oj}^{'} \prec X_{oj}$) si et seulement si $C_j^{'} \leq C_j$ et $\forall\, l \in [0, L]\ T_j^{l'} \leq T_j^l$ et $(R_j^{'}, C_j^{'}) \neq (R_j, C_j)$.

Desrochers [47] utilise une extension du principe d'optimalité de Bellman selon laquelle il suffit de conserver les étiquettes non-dominées.

Cet algorithme permet de trouver le plus court chemin non-élémentaire dans un graphe en respect des contraintes de ressources. Lorsque l'on a dans le graphe des coûts négatifs (ce qui est fréquent avec la technique de génération de colonnes) il est possible que les chemins optimaux ne soient pas élémentaires. Dans notre cas la présence des fenêtres de visite va limiter voire prohiber la présence de cycles de coûts négatifs. Pour autant nous avons décidé malgré la présence de fenêtres de visite dans notre problème d'utiliser l'extension proposée par Feillet *et al.* pour le cas élémentaire [60] qui interdit d'étendre un label à un nœud lorsqu'il a déjà été visité. En effet les fenêtres de visite utilisées sont dans certaines instances larges, la présence de cycle négatif peut donc être fréquente. De plus cette procédure, bien que plus consommatrice en temps de calcul, permet généralement de générer des meilleures colonnes.

Application à notre problème

Dans notre cas les ressources sont le temps et la capacité libre dans le véhicule. Ainsi, on s'assure que les deux contraintes sont respectées : respect des fenêtres de visite et respect de la capacité des véhicules. Le coût correspond à la somme des coûts des arcs empruntés par le chemin dans le graphe présenté en 6.33. Nous associons donc à chaque chemin R_j débutant au dépôt et arrivant au nœud j une étiquette $E_j = (C_j, T_j, L_j)$ où C_j est le coût réduit associé au chemin, T_j est l'instant d'arrivée du véhicule au nœud j et L_j la charge dans le véhicule à son arrivée en j. Pour expliquer la règle de dominance que nous utilisons prenons deux étiquettes E_j et $E_j^{'}$ associées respectivement aux chemins R_j et $R_j^{'}$ qui partent du dépôt et arrivent au site j. L'étiquette E_j domine ($\prec$) $E_j^{'}$ si tous les éléments de l'étiquette E_j sont inférieurs ou égal aux éléments de l'étiquette $E_j^{'}$.
$E_j \prec E_j^{'}$ si et seulement si :

- $C_j \leq C_j^{'}$;
- $T_j \leq T_j^{'}$;
- $L_j \leq L_j^{'}$.

Les prétraitements

Les prétraitements au sein du graphe du sous-problème permettent de réduire la combinatoire et ainsi améliorer les temps d'obtention d'une solution.

Dans la littérature nous avons trouvé plusieurs manières de réduire la taille du graphe pour le sous-problème en effectuant des éliminations d'arcs "inutiles". Citons par exemple les travaux de Rousseau *et al.* (2002, [132] et [133]). Nous n'avons pas pu les mettre en œuvre dans nos résolutions car l'inégalité triangulaire doit être respectée pour pouvoir les appliquer. Or, du fait du mélange de collectes et livraisons nous ne pouvons pas dire que l'inégalité triangulaire soit respectée pour la ressource correspondant à la charge dans le véhicule.

Par contre nous interdisons les arcs entre les sites dont la somme de leur charge fait dépasser la capacité des véhicules. Nous interdisons également les arcs entre deux sites dont les fenêtres de visite sont incompatibles. Ainsi, j ne peut pas succéder à i si $a_i + s_i + t_{ij} > b_j$.

6.2.5 La résolution du sous-problème par la méthode taboue

Dans le but de trouver une solution plus rapidement que par la programmation dynamique nous avons utilisé une méthode taboue. Cette méthode fait office de préliminaire pour la programmation dynamique, en effet lorsque la méthode taboue ne trouve plus de colonnes de coût réduit négatif la programmation dynamique prend le relais dans la recherche de nouvelles colonnes. La méthode taboue a été inventée par Glover (1989, [73] et 1990 [74]).

Le principe de la recherche taboue est très simple. Il consiste à partir d'une solution initiale s_0, puis lors

d'un processus itératif, chaque solution s_i est remplacée par la meilleure solution de son voisinage $V(s_i)$. Ce processus itératif est stoppé au bout de m itérations ou si h itérations ont été effectuées sans amélioration de la solution courante (m et h étant des paramètres de la résolution). Une liste appelée *"liste taboue"* retient les n derniers mouvements effectués afin d'interdire les mouvements inverses.

Le voisinage

Lors de l'exploration du voisinage d'une solution, quatre opérations sont possibles. Nous pouvons effectuer :
- un retrait d'un site de la solution ;
- un ajout d'un site dans la solution ;
- un échange de deux sites dans la solution ;
- un déplacement d'un site dans la solution.

La solution de départ

Nous commençons notre résolution grâce à la méthode taboue par une solution visitant un site i et effectuant sa livraison et sa collecte. Nous avons ainsi la tournée suivante : dépôt → livraison de la totalité de la demande du site i → collecte de la totalité de la demande du site i → dépôt. Si à partir de cette solution de départ aucune colonne améliorante n'est trouvée nous recommençons le processus en partant cette fois-ci d'une nouvelle solution de départ avec un autre site satisfait.

Les paramètres de notre résolution par méthode taboue

Les différents paramètres utilisés pour la résolution du plus court chemin pour le sous-problème que nous considérons sont :
- taille de la liste taboue : 15 ;
- nombre maximum d'itérations effectués : 10000 ;
- nombre maximum d'itérations sans amélioration : 500.

Toutes les colonnes dont le coût réduit est négatif qui sont trouvées au cours de la résolution de la méthode taboue sont ensuite ajoutées à l'ensemble des colonnes déjà présentent pour la résolution du programme maître.

6.2.6 La résolution du sous-problème par des techniques de programmation par contraintes

Pour la résolution du sous problème par des techniques de programmation par contraintes nous nous sommes inspirés des travaux réalisés par Rousseau *et al.* (2002, [132] et 2004, [133]).

Les variables et leur domaine

Pour résoudre le sous problème dans le cadre de la résolution du problème en juste à temps par les techniques de génération de colonnes, nous utilisons les variables suivantes :
- S_i : variable représentant le successeur du nœud i, $S = \{S_i\} \; \forall \; i \in I$;
- In_i : variable binaire égale à 1 si le site i est dans le chemin construit et 0 sinon ;
- t_i : variable représentant l'instant de visite du site i ;
- la variable AC_i : représente la capacité libre du véhicule avant son arrivée en i ;
- la variable DC_i : représente la capacité libre du véhicule après son passage en i ;
- la variable MAC_i : représente le minimum des capacités libres du véhicule à son arrivée chez tous les prédécesseurs de i, i compris. Si P_i représente l'ensemble des sites visités avant le site i alors $MAC_i = \min_{P_i \cup i}(AC_i)$;
- la variable MDC_i : représente le minimum des capacités libres du véhicule après son passage chez tous les successeurs de i, i compris. Si S_i représente l'ensemble des sites visitées après le site i alors $MDC_i = \min_{S_i \cup i}(DC_i)$;
- $PdtLiv$ représente l'ensemble des produits qui sont livrés par la tournée.

Les variables : AC_i, DC_i, MAC_i et MDC_i ont pour domaine de définition : $[0, CapaVeh]$. La variable S_i a pour domaine de définition l'ensemble des sites auxquels sont enlevés les sites interdits du fait de la typologie des sites considérés. In_i est une variable binaire. t_i quant à lui, a pour domaine de définition $[a_i, b_i]$ où a_i et b_i représentent respectivement la borne inférieure et supérieure de la fenêtre de visite du site i. La variable AC du dépôt est égale à la capacité totale du véhicule notée $capaVeh$, la variable DC du dépôt est égale à $capaVeh - PdtLiv$. Chaque variable DC est égale à la variable AC de son successeur. Pour vérifier si une insertion d'un site k de livraison entre le site i et j est possible (uniquement vis à vis de la contrainte de capacité des véhicules) il suffit de vérifier que la quantité $qtePdt_k$ de produits livrée au site k est telle que $qtePdt_k \leq MAC_i$. De même, pour vérifier si une insertion d'un site k de collecte entre le site i et j est possible il suffit de vérifier que la quantité $qtePdt_k$ de produits collectée au site k respecte $qtePdt_k \leq MDC_j$

Les contraintes

- $S_i = i \leftrightarrow In_i = 0$. Il s'agit ici d'une contrainte dite de *channeling* ;
- $AllDiff(S)$, tous les successeurs doivent être différents, afin d'assurer la conservation du flux ;
- $NoSubTour(S)$, assure qu'aucun sous-tour n'est fait par l'affectation des variables successeurs (cette contrainte est issue des travaux de Rousseau *et al.* (2002) [132]) ;
- $S_i = j \rightarrow t_i + t_{ij} + s_i \leq t_j$, assure le respect des fenêtres de visite ;
- $S_i = j \& In_i = 1 \rightarrow DC_j = DC_i - a_i^1 + a_i^2$, assure la cohérence dans le chargement du véhicule.

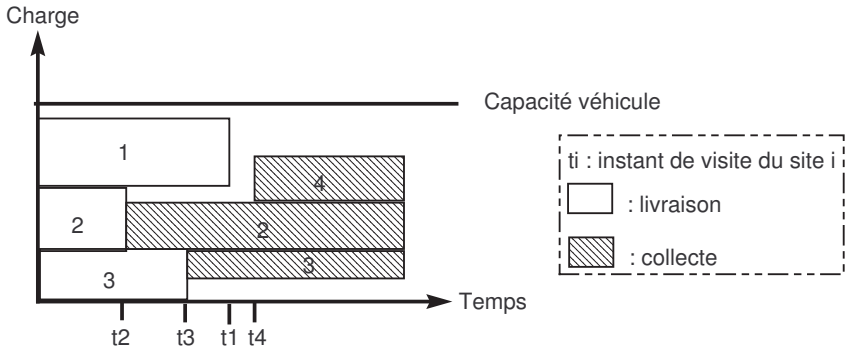
Les contraintes globales utilisées et leurs explications

Afin d'accélérer la recherche nous avons remplacé les différentes variables AC, DC, MAC, MDC et $PdtLiv$ et utilisé la contrainte globale *Cumulative(ressource, listeTaches, capacité)*. Celle-ci permet sur une ressource de maintenir l'ordonnancement des tâches de façon à ce que ces tâches soient toujours en dessous de la capacité de la ressource. Dans notre cas d'utilisation, chaque tâche de livraison qui est dans la tournée, représentant le plus court chemin, représente une tâche dont le début est l'instant 0. En effet, toutes les livraisons sont chargées depuis le nœud de départ. La fin des tâches de livraison s'effectue à l'instant de visite du site concerné. De même, pour les sites de collecte pris en compte dans la tournée construite nous créons une tâche de collecte dont le début est l'instant de visite du site de collecte considéré et sa fin est l'instant de visite du dépôt final, soit la fin de la tournée. Ainsi, les durées des tâches sont les suivantes :

- durée de la tâche associée à un site de livraison présent dans la tournée construite : (instant de visite du site - instant de départ du véhicule du dépôt) ;
- durée de la tâche associée à un site de collecte présent dans la tournée construite : (instant de fin de la tournée - instant de visite du site) ;
- durée de la tâche associée à n'importe quel site non présent dans la tournée : 0.

La figure 6.3 représente l'évolution de la charge dans le véhicule comme cela est fait avec la contrainte *Cumulative*.

Figure 6.3 *Évolution de la charge dans le véhicule au cours d'une tournée*



6.2.7 La solution initiale

Pour la résolution, d'une part, par la technique de génération de colonnes et, d'autre part, par la méthode taboue, nous avons besoin d'une solution initiale. Pour cela, nous créons une tournée par site physique (collecte et livraison). Ainsi la tournée i partira du dépôt visitera le site i pour effectuer la livraison et la collecte des quantités totales de ce site et reviendra au dépôt.

6.3 La résolution du problème avec gestion des stocks

6.3.1 La modélisation

Dans cette seconde modélisation nous considérons que les sites peuvent être livrés en avance de leur demande, moyennant un coût appelé coût de stockage. Nous allons ainsi résoudre notre problème dans sa version avec gestion des stocks. Pour écrire le modèle de partitionnement correspondant à ce problème avec gestion des stocks nous allons utiliser les mêmes notations que celles introduites dans le modèle en juste à temps (*cf.* 6.2), nous allons seulement rajouter la constante :

– cs : qui correspond au coût de stockage.

$$\text{Min :} \qquad z = \sum_{r\in\Omega}\sum_{t\in[1,T]} c_r x_{rt} + cs \sum_{t\in[1,T]}\sum_{i\in I}[(\sum_{r\in\Omega} a^1_{ir}x_{rt}) - d_{it}] \tag{6.42}$$

Sous Contraintes :

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} x_{rk}a^1_{ir} \geq \sum_{k=1}^{t} d_{ik} \tag{6.43}$$

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} x_{rt}a^2_{ir} \leq M_{ti} \tag{6.44}$$

$$\forall\, i \in I \quad \forall\, t \in [1,T] \quad \sum_{r\in\Omega}\sum_{k=1}^{t} A_{ir}x_{rk} \leq S'_{it} \tag{6.45}$$

$$\forall\, t \in [1,T] \qquad \sum_{r\in\Omega} x_{rt} \leq V \tag{6.46}$$

$$x_{rt} \in \aleph \tag{6.47}$$

L'objectif est de minimiser les coûts de routage et de stockage sous les contraintes suivantes :

– Contrainte 6.43 : Respect de la demande de chaque site pour chaque jour ;
– Contrainte 6.44 : Respect de la quantité possible à collecter de chaque site pour chaque jour ;
– Contrainte 6.45 : Respect de la capacité de stockage de chaque site ;
– Contrainte 6.46 : Respect de la capacité de la flotte de véhicules.

Introduisons la variable $D_{it} = \sum_{k=1}^{t} d_{it} \quad \forall i \in I\, \forall t \in T.$

Nous résumons dans le tableau 6.2 le schéma général de la matrice des contraintes correspondant au problème maître.

Associons à la contrainte 6.43 la variable duale μ_{it}, à la contrainte 6.44 la variable duale σ_{it}, à la contrainte 6.45 : β_{it} et à la contrainte 6.46 la variable duale α_t.

TAB. 6.2 – Tableau de la matrice des contraintes du problème maître du problème avec gestion des stocks

			Tournées																				
			jour 1					**jour 2**					**…**	**jour T**							**type**	**second**	
			1	**…**	**r**	**…**	**w**	**1**	**…**	**r**	**…**	**w**		**1**	**…**	**r**	**…**	**w**			**contrainte**	**membre**	**Duales**
	i=1	t=1	a^1_{11}	…	a^1_{1r}	…	a^1_{1w}						…								≥	D_{11}	μ_{11}
I * T		t=2	a^1_{11}	…	a^1_{1r}	…	a^1_{1w}	a^1_{11}	…	a^1_{1r}	…	a^1_{1w}									≥	D_{12}	μ_{12}
contraintes		…																			…	…	…
respect		t=T	a^1_{11}	…	a^1_{1r}	…	a^1_{1w}	a^1_{11}	…	a^1_{1r}	…	a^1_{1w}		a^1_{11}	…	a^1_{1r}	…	a^1_{1w}			≥	D_{1T}	μ_{1T}
demande	i=2	t=1	a^1_{21}	…	a^1_{2r}	…	a^1_{2w}														≥	D_{21}	μ_{21}
	…	…	…	…	…	…	…	…	…	…	…	…									…	…	…
	i =1	t=1	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}														≤	M_{11}	σ_{11}
I * T		t=2	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}									≤	M_{12}	σ_{12}
contraintes		…											…								…	…	…
respect		t=T	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}	a^2_{11}	…	a^2_{1r}	…	a^2_{1w}		a^2_{11}	…	a^2_{1r}	…	a^2_{1w}			≤	M_{1T}	σ_{1T}
quantité	i=2	t=1	a^2_{21}	…	a^2_{2r}	…	a^2_{2w}														≤	M_{21}	σ_{21}
collectée	…	…																			…	…	…
	i =1	t=1	A_{11}	…	A_{1r}	…	A_{1w}														≤	S'_{11}	β_{11}
I * T		t=2	A_{11}	…	A_{1r}	…	A_{1w}	A_{11}	…	A_{1r}	…	A_{1w}									≤	S'_{12}	β_{12}
contraintes		…											…								…	…	…
respect		t=T	A_{11}	…	A_{1r}	…	A_{1w}	A_{11}	…	A_{1r}	…	A_{1w}		A_{11}	…	A_{1r}	…	A_{1w}			≤	S'_{1T}	β_{1T}
capacité	i=2	t=1	A_{21}	…	A_{2r}	…	A_{2w}														≤	S'_{21}	β_{21}
stockage	…	…																			…	…	…
T		t=1	1	…	1	…	1														≤	V	α_1
contraintes		t=2						1	…	1	…	1									≤	V	α_2
respect		…											…								…	…	…
flotte		t=T												1	…	1	…	1			≤	V	α_T

Nous pouvons alors écrire le dual du problème ainsi :

$$\text{Max :} \qquad z' = \sum_{i\in I}\sum_{t\in[1,T]} D_{it}\mu_{it} - \sum_{i\in I}\sum_{t\in[1,T]} M_{ti}\sigma_{it} - \sum_{i\in I}\sum_{t\in[1,T]} S'_{it}\beta_{it} - \sum_{t\in[1,T]} V\alpha_t \quad (6.48)$$

Sous Contraintes :

$$\forall\, r \in \Omega\, \forall\, t \in [1,T] \quad \sum_{i\in I}\sum_{k=t}^{T} a^1_{ir}\mu_{ik} - \sum_{i\in I}\sum_{k=t}^{T} a^2_{ir}\sigma_{ik} - \sum_{i\in I}\sum_{k=t}^{T} A_{ir}\beta_{ik} - \alpha_t \leq c_r + cs\sum_{i\in I} a^1_{ir} \quad (6.49)$$

$$\forall\, i,t \qquad \mu_{it} \geq 0 \quad (6.50)$$

$$\forall\, i,t \qquad \sigma_{it} \geq 0 \quad (6.51)$$

$$\forall\, i,t \qquad \beta_{it} \geq 0 \quad (6.52)$$

$$\forall\, t \qquad \alpha_t \geq 0 \quad (6.53)$$

6.3.2 La description du sous-problème

Le sous-problème correspondant à cette deuxième version de résolution, ressemble à celui décrit en 6.2.2. Nous considérons, $G = (V', A)$ le graphe où V' représente l'ensemble des nœuds de notre problème et A l'ensemble des arcs réalisables. Comme précédemment, deux actions sont possibles dans chaque site : livraison de produits neufs et collecte de matériaux (palettes vides et retours de produits). C'est pourquoi un site est divisé en deux pour séparer sa partie collecte de sa partie livraison, ainsi une tournée peut passer plusieurs fois par un même site réel (une fois pour livrer et une autre fois pour collecter) mais une seule fois par site pour chaque action. De plus la quantité livrée et collectée n'est pas déterminée par avance. Notons P_{it} l'ensemble des valeurs possibles pour la quantité de livraison d'une tournée au site i le jour t. Cet ensemble est quant à lui plus grand que celui considéré en 6.2.2 car il comprend les valeurs de livraison possibles pour les jours suivants. En effet, nous considérons ici que l'on peut livrer avec avance sous contrainte d'un coût de stockage donc les quantités possibles pour les livraisons sur un jour sont agrémentées des quantités possibles pour les livraisons sur les jours ultérieurs. Nous avons également, C_{it} l'ensemble des valeurs possibles pour la quantité collectée d'une tournée au site i le jour t. Pour chaque site de livraison nous avons donc autant de nœuds de livraison que de quantité possible. Il en est de même pour les sites de collectes et leur déclinaison en nœuds de collectes selon les quantités. Ainsi dans notre graphe nous avons $2 + \sum_{i\in I, t\in T}(C_{it} + P_{it})$ nœuds. Soit x^r_{ij} paramètre binaire indiquant si l'arc (i,j) est emprunté par la tournée r. Les nœuds i et j correspondent à un site auquel on lui associe grâce à son numéro : une action (collecte ou livraison) et une quantité. c_{ij} est le coût associé à cet arc. Les graphiques 6.1 et 6.2 représentent les graphes dans lesquels nous travaillons pour la résolution du sous-problème.

Notons cs_i le coût de stockage engendré par le site i. Si le site i correspond à une quantité i d'une tournée. On peut alors exprimer le coût réel d'une tournée ainsi :

$$\forall r \in \Omega \quad \sum_{i\in A}\sum_{j\in A} x^r_{ij}c_{ij} + CF = c_r$$

Notons, a^1_j la quantité livrée au nœud j et a^2_j la quantité collectée au nœud j (dans un nœud on ne peut faire qu'une seule action si $a^2_j > 0$ alors $a^1_j = 0$ et vice versa). Pour chaque jour t nous pouvons créer un graphe G^t où les différents coûts vont être répartis en 5 types comme suit (les coûts sont portés par l'arc entrant des sites) :

- Pour tous les arcs (i,j) où i est le dépôt de départ et j est un nœud de livraison le coût associé est : $c_{ij} + CF + cs \times a^1_j - \sum_{k=t}^{T} a^1_j\mu^*_{jk} + \sum_{k=t}^{T} a^1_j\beta^*_{jk}$;
- Pour tous les arcs (i,j), où i est le dépôt de départ et j un nœud de collecte le coût associé est de : $c_{ij} + CF + \sum_{k=t}^{T} a^2_j\sigma^*_{jk} - \sum_{k=t}^{T} a^2_j\beta^*_{jk}$;
- Pour tous les arcs (i,j) avec i un nœud autre que le dépôt de départ ou d'arrivée et j un nœud de livraison le coût est : $c_{ij} + cs \times a^1_j - \sum_{k=t}^{T} a^1_j\mu^*_{jk} + \sum_{k=t}^{T} a^1_j\beta^*_{jk}$;

– Pour tous les arcs (i, j) avec i un nœud autre que le dépôt de départ ou d'arrivée et j un nœud de collecte le coût est : $c_{ij} + \sum_{k=t}^{T} a_j^2 \sigma_{jk}^* - \sum_{k=t}^{T} a_j^2 \beta_{jtk}^*$;
– Pour tous les arcs (i, j) avec i un nœud autre que le dépôt de départ ou d'arrivée et j le dépôt d'arrivée le coût est : $c_{ij} + \alpha_t^*$.

On peut alors formuler le sous-problème, caractérisant une tournée pour la journée t, ainsi :

$$\text{Min :} \quad \sum_{i\in A}\sum_{j\in A}(c_{ij} + csa_j^1)x_{ij} - \sum_{i\in A}\sum_{j\in A}\sum_{k=t}^{T}(a_j^1\mu_{jk}^* - a_j^2\sigma_{jk}^* - (a_j^1 - a_j^2)\beta_{jk}^*)x_{ij} - \alpha_t^* \quad (6.54)$$

Sous Contraintes :

$$\forall\, j \in A \qquad \sum_{i\in A} x_{ij} - \sum_{i\in A} x_{ji} = 0 \quad (6.55)$$

$$\sum_{i\in A} x_{0i} + \sum_{i\in A} x_{i0} = 2 \quad (6.56)$$

$$\forall\, i \in A,\ j \in A \qquad CL_j x_{ij} = CL_i x_{ij} - a_j^1 + a_j^2 \quad (6.57)$$

$$\forall\, i \in A \qquad 0 \leq CL_i \leq CapaVeh \quad (6.58)$$

$$CL_0 = \sum_{i\in A} a_i^1 \quad (6.59)$$

$$\forall\, i \in A,\ j \in A \qquad t_i + s_i + t_{ij} - (1 - x_{ij})M \leq t_j \quad (6.60)$$

$$\forall\, i \in A \qquad a_i \leq t_i \leq b_i \quad (6.61)$$

$$\forall\, i \in A,\ j \in A \qquad x_{ij} \in \{0, 1\} \quad (6.62)$$

L'objectif est ici de trouver des tournées dont le coût réduit est négatif (6.54). La tournée générée doit respecter la conservation des flux (6.55). La tournée doit débuter et terminer au dépôt (6.56). La capacité du véhicule doit être respectée (6.57, 6.58, 6.59). Les tournées doivent respecter les fenêtres de visites des sites (6.60, 6.61).

Les méthodes de résolution de sous-problème associé à notre problème de construction de tournées avec gestion des stocks sont les mêmes que celles utilisées pour la version du problème en juste à temps. En effet, hormis la fonction objectif du sous-problème qui change toutes les contraintes restent les mêmes.

6.4 La dégénérescence

La génération de colonnes est guidée par les coûts réduits associés aux colonnes dépendants des variables duales associées au problème. Dans certains cas plusieurs combinaisons de valeurs des variables duales mènent à une même valeur de la fonction objective tant primale que duale. Il s'agit alors de dégénérescence. Avant de mettre en place des techniques de stabilisation qui permettraient d'enrailler ce problème, nous avons souhaité tester notre problème sur l'ensemble des instances de façon à voir si ce problème était présent. Les tests effectués n'ont pas mis en évidence un tel phénomène. Malgré tout si pour de plus grandes instances une dégénérescence était observée une technique de stabilisation telle que celle proposée par Rousseau *et al.* (2007) [131] pourrait être mise en œuvre.

6.5 Le *branch and bound*

La résolution exacte du problème suppose le développement d'une méthode de *Branch and Price* (Barnhart *et al.*, 1998 [7] et Vanderbeck, 2000 [159]). Dans le cadre de cette thèse, nous nous limiterons à la résolution par une méthode de *Branch and Bound* [102] classique, limitée aux colonnes générées pour la résolution de la relaxation linéaire du problème au premier nœud. Pour cela, nous utilisons le *Branch and Bound* proposé par Xpress. La solution proposée sera donc une borne supérieure de la solution de notre problème.

De plus, la solution obtenue grâce à la génération de colonnes correspond à une borne supérieure de la solution du problème considéré lors de la résolution grâce à des méthodes heuristiques ou métaheuristiques. En effet, dans les résolutions heuristiques et métaheuristiques, toutes les quantités de livraison sont envisageables, alors que dans une résolution par génération de colonnes le nombre de possibilités de quantité de livraison est restreint. Le résultat obtenu sert donc de borne supérieure à l'optimum du problème.

6.6 Les résultats

Nous allons ici donner les résultats obtenus grâce aux différentes versions de la génération de colonnes suivie du *Branch and Bound*. Nous avons arrêté l'exécution lorsque cela dépassait une heure. Les différents tests effectués avec la génération de colonnes suivie d'un *Branch and Bound* sont effectués sur des instances plus petites que celles utilisées précédemment. Nous utilisons les instances composées de 5 et 6 sites dont les caractéristiques ont été données dans le chapitre 3.

6.6.1 En juste à temps

	5 sites				6 sites			
	ProgDyn	**Taboue**	**Ppc**	**PpcCumu**	**ProgDyn**	**Taboue**	**Ppc**	**PpcCumu**
Nombre problèmes résolus	93/108 86%	100/108 93%	70/108 65%	49/108 45%	71/96 74 %	61/96 64 %	34/96 35 %	28/96 29%
Temps moyen obtention solution relaxée	30 s	14 s	727 s	144 s	475 s	61 s	60 s	343 s
Nombre solutions entière obtenue	93/108 86%	100/108 93%	70/108 65%	49/108 45%	71/96 74%	61/96 64%	71/96 74 %	28/96 30%
Temps moyen obtention solution entière	8 s	0,1 s	1,7 s	35 s	244 s	24 s	8 s	185 s
Nombre colonnes	768	547	163	289	924	728	196	332
Nombre appels sous-problème	4,3	5,6	9,6	9,4	4,6	6,2	11	8,8
Nombre sol entières = sol relaxées	79/93 85 %	55/100 55 %	31/70 44 %	21/49 43%	54/71 76 %	29/61 48 %	9/34 26 %	8/28 29 %
$\neq$ sol entières sol relaxées	0,3 %	1,3 %	5%	4 %	1,7%	3 %	11 %	4,5%

TAB. 6.3 – Comparaison des résultats obtenus par la génération de colonnes selon les méthodes de résolution et les tailles d'instances en juste à temps

Le tableau 6.3 donne les résultats obtenus pour les instances à 5 sites (108 instances) et les instances à 6 sites (96 instances) dont les spécifications ont été données dans le chapitre 3. Pour chaque taille d'instance nous trouvons tour à tour les résultats obtenus pour chaque méthode de résolution du sous-problème (ProgDyn pour la programmation dynamique, Taboue pour la méthode taboue, Ppc pour la programmation par contraintes sans contrainte globale et PpcCumu pour la programmation par contraintes avec contrainte globale). Nous donnons à chaque fois le nombre d'instances résolues en moins d'une heure, le temps moyen d'obtention de la solution relaxée, le nombre de solution entière trouvée grâce à la méthode, le temps moyen d'obtention de la solution entière, le nombre de colonnes générées, le nombre d'appels au sous-problème, le nombre de fois où nous obtenons la solution optimale du problème et l'écart moyen entre la solution entière obtenue et la solution relaxée. Quelque soit la méthode de résolution du sous-problème, nous constatons dans ce tableau que plus la taille des instances augmente plus il est difficile de trouver une solution au problème en moins d'une heure. La méthode du *Branch and Bound* pour trouver une solution entière au problème semble être adaptée, puisqu'elle trouve en général des résultats très proches de ceux obtenus pour la borne inférieure grâce à la relaxation. L'utilisation d'une technique de *Branch and Price* permettrait quant à elle de toujours trouver la solution optimale du problème mais dans des temps de calcul plus long. Comme il a été remarqué dans le chapitre 5 l'emploi de la programmation par contraintes est plus difficile lorsqu'il y a des fenêtres de temps larges. C'est ce qui explique la différence au niveau du nombre de problèmes résolus. L'utilisation de la programmation par contraintes permet

de trouver la solution relaxée en générant beaucoup moins de colonnes que les autres méthodes. Par contre ceci est un désavantage lors de la recherche d'une solution entière par *Branch and Bound* ce qui explique l'écart plus important par rapport aux autres méthodes entre la solution relaxée et la solution entière. Une complémentarité dans les résultats est observée, en effet, en ce qui concerne les instances à 5 sites la programmation dynamique et la méthode taboue ne trouvent pas de résultats en moins d'une heure sur les instances : C101J, C101M, C102J, C102M, RC101J, RC101M, RC101N et RC102C. Ainsi 7,5 % des instances ne trouvent un résultat en moins d'une heure que grâce à l'emploi de la programmation par contraintes. L'utilisation de la programmation par contraintes semble donc être propice lorsque les instances sont en cluster, pas de fenêtre de visite large et de même répartition que les catégories J, M et N (*cf.* chapitre 3). Lorsqu'une instance n'est pas résolue en moins d'une heure cela est dû à la recherche de la solution entière. Pour toutes les instances il est possible selon les méthodes de trouver une solution relaxée au problème en moins d'une heure. L'utilisation de la programmation par contraintes suivie par la programmation dynamique (si un certain temps est dépassé sans obtention de solution) permettrait ici de résoudre 94 % des instances à 5 sites.

6.6.2 Avec gestion des stocks

	5 sites				6 sites			
	ProgDyn	**Taboue**	**Ppc**	**PpcCumu**	**ProgDyn**	**Taboue**	**Ppc**	**PpcCumu**
Nombre problèmes résolus	108/108 100%	107/108 99%	89/108 82 %	58/108 54 %	93/96 96%	93/96 96%	62/96 65 %	42/96 44 %
Temps moyen obtention solution relaxée	4,5 s	3,9 s	20 s	35 s	88 s	53 s	85 s	50 s
Nombre solutions entière obtenue	24/108 22 %	23/108 22 %	6/108 6 %	0/108 0 %	21/96 22 %	21/96 22 %	4/96 6 %	4/96 4 %
Temps moyen obtention solution entière	123 s	0,1 s	158 s	-	0,3 s	1,6 s	115 s	403 s
Nombre colonnes	292	238	129	125	368	304	157	231
Nombre appels sous-problème	1,8	2,1	1,8	1	1,9	2,4	1,8	2,7
Nombre sol entières = sol relaxées	22/24 92 %	23/23 100 %	6/6 100 %	- -	18/21 86 %	21/21 100 %	2/4 50 %	3/4 75 %
$\neq$ sol entières sol relaxées	0,4 %	0 %	0 %	-	0,004 %	0 %	3 %	0,02 %

TAB. 6.4 – Comparaison des résultats obtenus par la génération de colonnes selon les méthodes de résolution et les tailles d'instances avec gestion des stocks

Le tableau 6.4 donne les résultats obtenus grâce à la technique de génération de colonnes suivie d'un *Branch and Bound* sur le problème dans sa version avec gestion des stocks. Les instances testées sont celles expliquées dans le chapitre 3. Pour chaque instance nous donnons tour à tour les mêmes informations que celles données pour les résultats en juste à temps. Nous constatons dans ce tableau que l'obtention d'une solution entière est rendue difficile par la mise en place de l'option de gestion des stocks. Les colonnes générées pour trouver la solution relaxée du problème ne permettent pas dans la majorité des cas de trouver une solution entière au problème non relaxé. C'est pourquoi dans ce cas une technique de *Branch and Price* serait préférable pour trouver une solution entière à tous les problèmes et ce malgré le faible écart trouvé entre la solution entière et la solution relaxée sur les instances résolues entièrement. Ceci constitue la principale perspective de l'étude du problème avec gestion des stocks et grâce à des méthodes exactes. On constate que la résolution par programmation par contraintes utilisant la contrainte cumulative résout moins d'instances en moins d'une heure. Par contre parmi les 42 instances résolues 25 instances (essentiellement dans les catégories L, M et N) sont des instances qui ne sont pas résolues par la programmation par contraintes sans contrainte globale. Ce qui fait qu'en cumulant les deux approches nous sommes en mesure de résoudre 90 % des instances en moins d'une heure avec une approche hybride.

6.7 Conclusions

Dans ce chapitre nous avons proposé une méthode et des algorithmes basés sur des méthodes exactes pour résoudre notre problème. Plusieurs techniques ont été utilisées pour résoudre le sous-problème de plus court chemin sous contraintes de ressources. Nous avons été contraints de tester nos méthodes sur des instances plus petites que pour les autres méthodes de résolution développées. De même, les demandes qui dans les méthodes de résolution heuristiques et métaheuristiques étaient divisibles à l'unité, n'ont pas pu être étudiées telles quelles avec la résolution basée sur des méthodes exactes. En effet, la multiplication du nombre de nœuds dans le graphe du sous-problème en fonction du nombre de demandes possibles par clients, augmente le temps de recherche d'une tournée améliorante.

Les résultats obtenus grâce aux diverses méthodes sont prometteurs et prouvent la complémentarité des méthodes issues de la recherche opérationnelle et celles issues de la programmation par contraintes. En effet, on constate que lorsque l'une des méthodes n'arrive pas à trouver une solution l'autre le fait.

Les différentes perspectives concernant la résolution basée sur les méthodes exactes résident essentiellement dans le gain de temps. Des techniques telles que l'utilisation de coupes, l'ajout d'inégalités valides, le calcul de bornes du problème, n'ont pas été suffisamment développées ici et pourraient permettre d'améliorer nos temps de résolution pour chacune des méthodes et plus particulièrement pour les techniques de programmation par contraintes. Ainsi les gains de temps et de mémoire permettraient de faire évoluer les paramètres de tests concernant la division des demandes ou de pouvoir traiter des instances avec plus de sites.

Afin de pouvoir comparer les résultats obtenus grâce aux méthodes développées il serait intéressant de les tester sur des problèmes plus répandus dans la littérature tels que le VRPTW.

Afin d'alléger la programmation par contraintes, la technique LDS (Limited Discrepancy Search) introduite par Harvey et Ginsberg [87] (de plus amples explications sont données dans la partie 5.2.2) peut être mise en place. Cette méthode permet de diminuer le nombre d'arcs explorés. En effet, lorsqu'un nœud est exploré au lieu d'étendre à tous les nœuds accessibles depuis celui-ci, on ne considère que les "meilleurs" nœuds c'est-à-dire ceux qui sont les plus proches en termes de coût réduit du nœud à étendre. Ainsi un premier paramètre de cette méthode est le nombre de nœuds que l'on s'autorise à parcourir. Le second paramètre représente le nombre de divergences que l'on autorise par rapport à la stratégie de sélection de nœuds. Cette méthode permet de réduire la combinatoire mais ne permet pas forcément de trouver la solution optimale.

Enfin une perspective serait de mettre en place une technique de *Branch and Price* pour le problème dans sa version avec gestion des stocks. Une éventuelle autre solution au problème de recherche de solution entière serait de ne pas ajouter que les colonnes de coûts négatifs à chaque résolution du sous-problème (par exemple ajouter les contraintes de coût inférieur à 10), ainsi l'on disposerait de plus de colonnes et l'on pourrait plus facilement trouver une solution entière au problème.

Troisième partie

Étude Comparative et Préconisations

Chapitre 7

Selon les catégories d'instances

Nous allons dans ce chapitre analyser les différents résultats obtenus selon les différentes catégories d'instances, selon les types de catégories de Solomon [140] (C1, C2, R1, R2, RC1 et RC2) et selon les différentes méthodes de résolution approchées que nous avons utilisées. Le but de ce chapitre est de déterminer selon la configuration des données quelle va être la meilleure méthode à utiliser pour obtenir le meilleur résultat approché. Durant les différents chapitres 4, 5 et 6 les résultats sont donnés en moyenne sur toutes les catégories d'instances confondues. Nous allons ici faire une analyse plus fine que celle faite dans la deuxième partie de cette thèse.

7.1 Les méthodes heuristiques

Dans cette partie nous allons comparer les résultats obtenus sur les différentes méthodes d'amélioration testées et sur les différentes combinaisons de méthodes détaillées dans la chapitre sur les méthodes de construction et améliorations (*cf.* chapitre 4).

7.1.1 Comparaison des méthodes d'amélioration

Le tableau 7.1 compare les différentes méthodes d'amélioration selon les catégories d'instances. Dans ce tableau le chiffre 1 correspond à la méthode d'amélioration ayant donné le meilleur coût moyen sur la catégorie d'instance traité, le chiffre 2 correspond à la méthode d'amélioration ayant donné le deuxième meilleur coût sur la catégorie traité *etc.* Nous constatons que pour toutes les catégories d'instances la meilleure combinaison "méthode de construction - méthode d'amélioration" est la méthode $MI + D_o$. La combinaison donnant le deuxième meilleur coût est $MI + A$. C'est à partir de la troisième meilleure méthode que des différences selon les types d'instances se révèlent.

Une entreprise qui voudrait n'utiliser qu'une méthode de construction suivie d'une méthode d'amélioration devrait donc utiliser $MI + D_o$ et $MI + A$ (ce qui implique une politique de gestion des stocks) peu importe la physionomie de son réseau.

7.1.2 Comparaison des méthodes de construction et améliorations

En juste à temps

Nous allons ici observer les différents résultats obtenus avec les méthodes de construction et améliorations dans la version en juste à temps du problème.

Nous allons ainsi voir toutes les méthodes expliquées dans le chapitre 4. Nous pouvons dans un premier temps comparer les résultats selon les types instances. Nous constatons que les instances A, B, C, F, J et K ont pour meilleure méthode de résolution par construction et améliorations : $MI + E_iE_oD_oD_iO^+$ et comme deuxième meilleure méthode : $MI + E_oD_oO^+$. Les instances D, E, G, H et I ont quant à elle pour meilleure méthode de construction et améliorations : $MI + E_oD_oO^+$ et comme deuxième meilleure méthode : $MI + E_iE_oD_oD_iO^+$. Pour une utilisation concrète le choix de la meilleure méthode de construction et améliorations se fera donc entre :

	MI						PMI					
	D_i	E_i	O	D_o	E_o	A	D_i	E_i	O	D_o	E_o	A
A	6	5	4	1	3	2	12	10	11	7	9	8
B	6	5	4	1	3	2	12	10	11	7	9	8
C	6	4	3	1	5	2	12	11	10	7	9	8
D	6	5	3	1	4	2	12	11	10	7	9	8
E	6	4	3	1	5	2	12	11	10	7	9	8
F	6	4	3	1	5	2	12	11	10	7	9	8
G	6	5	3	1	4	2	12	11	10	7	9	8
H	6	5	4	1	3	2	12	10	11	7	9	8
I	6	5	3	1	4	2	12	11	10	7	9	8
J	6	5	4	1	3	2	12	11	10	7	9	8
K	6	5	4	1	3	2	12	11	10	7	9	8

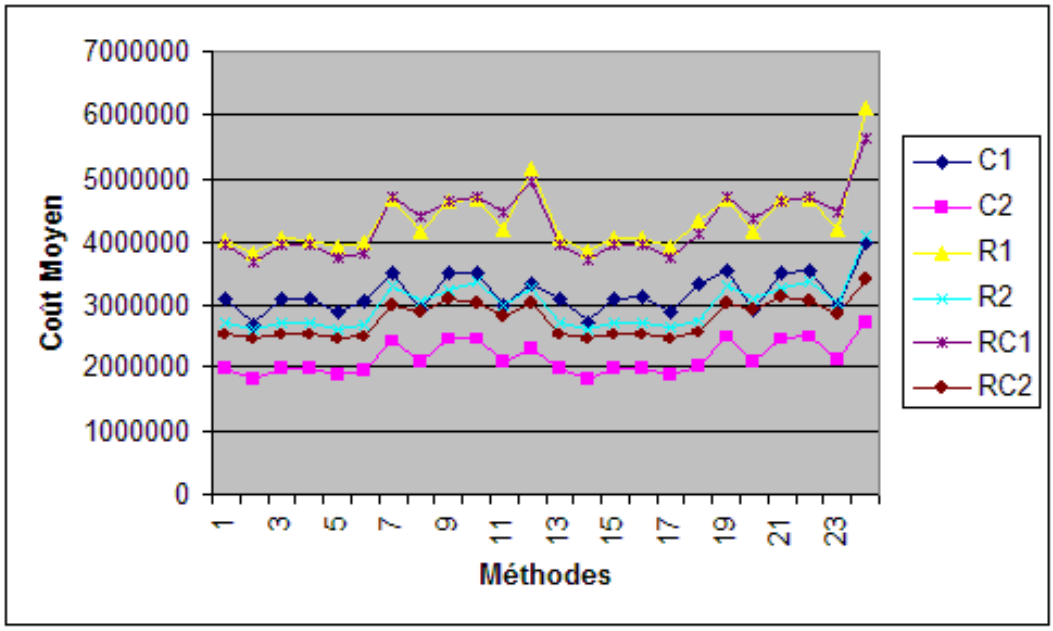
TAB. 7.1 – Comparaison des méthodes d'amélioration selon les catégories d'instances

$MI + E_oD_oO^+$ et $MI + E_iE_oD_oD_iO^+$. Aucune caractéristique remarquable entre les différentes catégories marchant mieux avec l'une ou l'autre des méthodes n'a pu être identifiées. Comme les deux méthodes sont toujours les deux premières l'une ou l'autre peuvent être considérées comme bonnes.

Dans un second temps nous allons détailler les résultats selon les séparations faites par Solomon [140] : C1, C2, R1, R2, RC1 et RC2. Une explication sur les spécialités des classes C1, C2, R1, R2, RC1 et RC2 est donnée en chapitre 3. Dans cette partie afin d'alléger le schéma nous noterons : $1 : \underrightarrow{MI + E_iE_oD_oD_iO}$; $2 : \underline{MI + E_iE_oD_oD_iO^+}$; $3 : MI + D_oD_iOE_iE_o$; $4 : MI + OE_iE_oD_oD_i$; $5 : \underrightarrow{MI + E_iE_oD_oD_iO}$; $6 : \underline{\underrightarrow{MI + E_iE_oD_oD_iO}}$; $7 : PMI + E_iE_oD_oD_iO$; $8 : PMI + E_iE_oD_oD_iO^+$; $9 : PMI + D_oD_iOE_iE_o$; $10 : PMI + OE_iE_oD_oD_i$; $11 : \overline{PMI + E_iE_oD_oD_iO}$; $12 : \underline{\overline{PMI + E_iE_oD_oD_iO}}$; $13 : MI + E_oD_oO$; $14 : MI + E_oD_oO^+$; $15 : MI + D_oOE_o$; $16 : MI + OE_oD_o$; $17 : \overline{MI + E_oD_oO}$; $18 : \underline{\overline{MI + E_oD_oO}}$; $19 : PMI + \underline{E_oD_oO}$; $20 : PMI + E_oD_oO^+$; $21 : PMI + D_oOE_o$; $22 : PMI + OE_oD_o$; $23 : \overline{PMI + E_oD_oO}$; $24 : \underline{\overline{PMI + E_oD_oO}}$.

Sur le graphique 7.1 nous voyons l'évolution des coûts moyens obtenus sur les types d'instances C1, C2, R1, R2, RC1 et RC2 dans la version en juste à temps du problème. Nous constatons que dans tous les cas la meilleure méthode est $MI + E_iE_oD_oD_iO^+$. Ainsi les positions des sites dans l'espace et les tailles des fenêtres de visite n'ont pas d'influence sur le choix de la méthode de résolution pour le problème en juste à temps.

Figure 7.1 *Comparaison des méthodes de résolution par construction et amélioration en juste à temps*



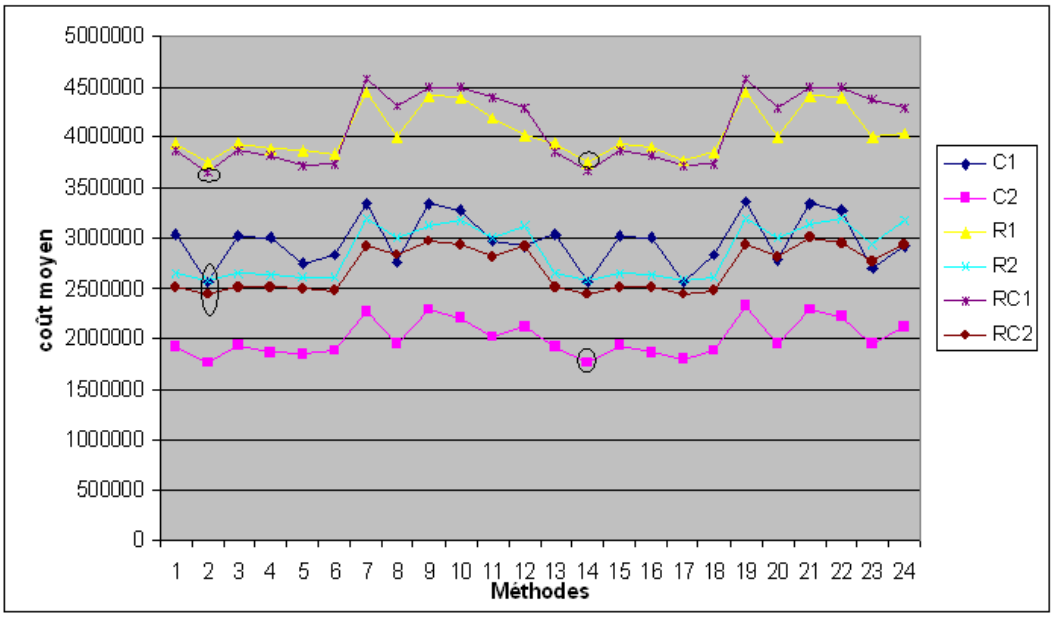
Avec gestion des stocks

Dans cette partie nous allons détailler les méthodes de résolution de construction et améliorations sur le problème dans sa version avec gestion des stocks.

Tout d'abord faisons une distinction selon les catégories d'instances. Les catégories B, C, E, F, J et K ont pour meilleure méthode : $MI + E_iE_oD_oD_iOA^+$ et pour meilleure deuxième méthode : $MI + E_oD_oOA^+$. Les instances A, D, G, H et I ont pour meilleure méthode : $MI + E_oD_oOA^+$ et pour deuxième meilleure méthode : $MI+E_iE_oD_oD_iOA^+$. Les catégories B, C, E, F, J et K étant principalement constituées de grands et très grands sites, nous déduisons qu'il vaut mieux utiliser la méthode $MI + E_iE_oD_oD_iOA^+$ en cas de nombreux grands sites dans la physionomie de l'instance et $MI + E_oD_oOA^+$ sinon.

Puis détaillons grâce au tableau 7.2 les résultats selon les types C1, C2, R1, R2, RC1 et RC2 nous allons utiliser les notations suivantes : 1 : $MI + E_iE_oD_oD_iOA$; 2 : $MI + E_iE_oD_oD_iOA^+$; 3 : $MI + D_oD_iOAE_iE_o$; 4 : $MI+OE_iE_oAD_oD_i$; 5 : $\overrightarrow{MI + E_iE_oD_oD_iOA}$; 6 : $\overrightarrow{MI + E_iE_oD_oD_iOA}$; 7 : $PMI+E_iE_oD_oD_iOA$; 8 : $PMI+E_iE_oD_oD_iOA^+$; 9 : $PMI+D_oD_iOAE_iE_o$; 10 : $PMI+OE_iE_oAD_oD_i$; 11 : $\overrightarrow{PMI + E_iE_oD_oD_iOA}$; 12 : $\overrightarrow{PMI + E_iE_oD_oD_iOA}$; 13 : $MI + E_oD_oOA$; 14 : $MI + E_oD_oOA^+$; 15 : $MI + D_oOAE_o$; 16 : $MI+OE_oAD_o$; 17 : $\overrightarrow{MI + E_oD_oOA}$; 18 : $\overrightarrow{MI + E_oD_oOA}$; 19 : $PMI+E_oD_oOA$; 20 : $PMI+E_oD_oOA^+$; 21 : $PMI+D_oOAE_o$; 22 : $PMI+OE_oAD_o$; 23 : $\overrightarrow{PMI + E_oD_oOA}$; 24 : $\overrightarrow{PMI + E_oD_oOA}$. Dans le tableau 7.2 la meilleure méthode est entourée. Pour les types C2 et R1 la meilleure méthode est : $MI+E_oD_oOA^+$. Pour les types C1, R2, RC1 et RC2 la meilleure méthode est : $MI + E_iE_oD_oD_iOA^+$. Ici aussi aucune distinction ne peut être faite selon les positions des sites et les tailles des fenêtres de visite dans les instances. En effet la méthode $MI + E_oD_oOA^+$ marche le mieux sur les instances C2 et R1 or ces deux types représentent les deux oppositions dans la typologie des instances. Il en est de même pour la méthode $MI + E_iE_oD_oD_iOA^+$.

Figure 7.2 *Comparaison des méthodes de résolution par construction et amélioration avec gestion des stocks*



7.1.3 Comparaison des différentes versions du GRASP

Dans cette partie nous détailler les résultats obtenus selon les types d'instances C1, C2, R1, R2, RC1 et RC2.

En juste à temps

La meilleure méthode de résolution pour toutes les catégories d'instances est la méthode GRASP classique avec une taille liste de candidats de 3. Voyons le détail selon les types (C1, C2, R1, R2, RC1 et RC2) dans le tableau 7.2. Chaque case donne les informations suivantes : "Class." si la meilleure méthode est de type classique, "Hybr." s'il est de type hybride. Le deuxième terme qui est donné représente la taille de la liste de candidats. Lorsqu'il s'agit d'une méthode hybride les renseignements suivants sont donnés dans cet ordre :

le nombre de divergence pour LDS, l'heuristique de choix des variables (MD pour MinDomain, DOD pour DomOverDeg, MC pour MostConstrained) et l'heuristique de choix des valeurs (ID pour IncreasingDomain, DD pour DecreasingDomain, R pour RandomIntValSelector). Les croisements des caractéristiques A-C1, B-RC1, I-RC1 semblent propices à l'utilisation de méthodes hybrides pour une résolution métaheuristique.

	A	B	C	D	E	F	G	H	I	J	K
C1	Hyb.-3 3-MD-ID	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3
C2	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3
R1	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3
R2	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3
RC1	Class. 3	Hyb.-3 3-DOD-ID	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Hyb.-3 3-MD-ID	Class. 3	Class. 3
RC2	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3	Class. 3

TAB. 7.2 – Comparaison des résultats sur les types C1, C2, R1, R2, RC1 et RC2 selon les différentes méthodes GRASP

Pour plus de détails nous pouvons voir le tableau 7.3 qui donne selon la catégorie d'instances le nombre de fois où chaque type de méthode (Construction et améliorations - C & A, GRASP classique et GRASP hybride) donne la meilleure solution obtenue toutes méthodes confondues. Pour rappel il y a 168 instances dans chaque catégorie. Une somme par catégorie supérieure à 168 signifie que plusieurs méthodes donnent la meilleure solution trouvée. Nous pouvons constater que les méthodes hybrides semblent ne pas beaucoup marcher sur les instances de type : C, E et F. Ces instances sont des instances avec majoritairement des très grands sites.

Avec gestion des stocks

En ce qui concerne les résultats obtenus pour le problème avec gestion des stocks nous constatons que pour tous les types d'instances (C1, C2, R1, R2, RC1 et RC2) les meilleurs résultats sont obtenus grâce à la méthode GRASP classique avec une taille de liste de 3. Ainsi pour un cas réel l'utilisation de cette métaheuristique pour un problème avec gestion des stocks semble propice.

Dans ce chapitre nous avons détaillé les résultats que nous avons obtenus selon les différentes catégories d'instances (A, B, C *etc.*) puis selon les différents types donnés dans les instances de Solomon [140] (C1, C2, R1 *etc.*). Nous avons pu constater qu'il n'existe pas de relation entre les résultats obtenus et les positions des sites

	C & A	Class	Hyb
A	2	151	19
B	4	152	17
C	0	163	5
D	0	157	12
E	0	163	5
F	1	160	9
G	3	157	12
H	3	158	11
I	2	148	20
J	3	150	18
K	0	147	21

TAB. 7.3 – Comparaison des meilleurs résultats obtenus en fonction des méthodes de résolution et des instances

au sein des instances ni avec les tailles des fenêtres de visites. Les résultats également ont été détaillé selon les catégories d'instances (selon les répartitions des sites de différentes tailles) ici aussi aucune distinction ne peut être faite selon les répartitions des tailles de sites au sein des instances.

Chapitre 8

Préconisations selon les applications

Dans le chapitre précédent nous avons étudié les différents résultats selon les caractéristiques "physiques" des instances : position des sites dans l'espace, taille des fenêtres de visite et répartition des différentes tailles de sites au sein des instances. Nous allons dans ce chapitre nous concentrer sur les caractéristiques "stratégiques" des instances : selon les coûts logistiques (coût fixe d'utilisation d'une tournée, coût au kilomètre et coût de stockage) et selon le choix ou non de pouvoir faire passer plusieurs véhicules pour satisfaire une seule demande.

8.1 Pertinence du partage de la demande selon les coûts de création et la physionomie de l'instance

Dans notre étude plusieurs véhicules peuvent livrer une même demande. Nous avons voulu tester ici quels seraient les résultats si nous avions fait l'hypothèse inverse.

8.1.1 Avec les méthodes de construction et améliorations

Pour cela, nous choisissons d'utiliser les deux meilleures méthodes identifiées précédemment : "$MI + E_iE_oD_oD_iOA^+$" pour les versions avec gestion des stocks et "$MI + E_oD_oO^+$" pour les versions en juste à temps. Nous faisons varier ici les coûts de création d'une tournée (coût fixe d'une tournée) mais nous conservons les mêmes valeurs que précédemment pour les coûts de stockage et les coûts de routage. Nous choisissons de tester les valeurs : 2000, 20000 et 200000 pour le coût de création d'une tournée (écrit CC dans le tableau). Dans

Version	Nombre Min avec Partage	Nombre Min sans Partage
CC = 2000 juste à temps	34 %	83 %
CC = 2000 avec gestion des stocks	59 %	74 %
CC = 20000 juste à temps	56 %	75 %
CC = 20000 avec gestion des stocks	59 %	71 %
CC = 200000 juste à temps	61 %	69 %
CC = 200000 avec gestion des stocks	64 %	66 %

TAB. 8.1 – Comparaison des différentes versions selon les coûts de création d'une tournée avec des méthodes de construction et améliorations

le tableau 8.1, nous donnons le nombre de fois où les meilleurs résultats sont obtenus selon les différents coûts de création d'une tournée (CC), selon l'hypothèse faite quant à l'avance ou non des livraisons et selon l'hypothèse faite sur le partage ou non des livraisons.

Nous constatons ici que l'hypothèse que nous avons choisie pour notre problème, qui est de partager la demande, n'est peut-être pas la meilleure pour cette configuration de coûts. Par contre, on se rend facilement compte que plus le coût de création d'une tournée augmente plus l'écart se réduit entre le nombre de meilleures solutions obtenues par l'une ou par l'autre des stratégies. Une explication à cette constatation est que le partage de la demande permet de diviser une demande en plusieurs parties afin de combler des tournées déjà constituées

au lieu de créer une nouvelle tournée. Par contre il faut pour effectuer cette opération que la création d'une tournée soit plus coûteuse que plusieurs visites du même site par plusieurs véhicules.

	A	B	C	D	E	F	G	H	I	J	K
CC = 2000 juste à temps	58%	66%	76%	68%	72%	68%	62%	63%	64%	64%	68%
CC = 20000 juste à temps	4%	45%	55%	52%	51%	54%	11%	32%	50%	41%	58%
CC = 200000 juste à temps	3%	45%	30%	47%	38%	43%	11%	35%	46%	44%	55%
CC = 2000 avec gestion des stocks	4%	42%	54%	48%	55%	49%	40%	27%	44%	37%	53%
CC = 20000 avec gestion des stocks	5%	41%	49%	48%	47%	49%	11%	30%	46%	36%	52%
CC = 200000 avec gestion des stocks	5%	40%	24%	43%	39%	36%	11%	33%	42%	42%	53%

TAB. 8.2 – Comparaison des répartitions de meilleures solutions obtenues grâce à la méthode de construction et améliorations dans la version sans partage de la demande selon les catégories d'instances

Dans le tableau 8.2 nous donnons le nombre de fois où la solution obtenue par la version avec non partage de la demande obtient de meilleurs résultats que celle autorisant le partage de la demande en fonction des différents coûts de création utilisés (les coûts de stockage et de routage restent identiques et sont égales à 10) et selon les catégories d'instances expliquées dans le chapitre 3. Nous constatons bien que l'option "demande partagée entre plusieurs véhicules" devient d'autant plus avantageuse que le coût de création d'une tournée augmente. Ce constat est d'autant plus flagrant sur les instances A et G. Ces deux types d'instances correspondent à des instances ayant majoritairement des petits sites. Il est donc très intéressant de mettre en place une politique de partage de la demande dans le cas où les sites sont de petites tailles.

Lorsque nous regardons transversalement les résultats obtenus pour les types d'instances de Solomon (C, R et RC) dans le tableau 8.3, nous constatons que le même phénomène se produit : plus le coût de création d'une tournée augmente plus l'option du partage de la demande est avantageuse. Nous constatons que ce phénomène est très marqué sur le type : RC1. Il s'agit ici des instances où certains sites sont placés en "cluster" et d'autres aléatoirement et ayant des fenêtres de visite serrées.

	C1	C2	R1	R2	RC1	RC2
CC = 2000 juste à temps	100%	100%	100%	88%	61%	94%
CC = 20000 juste à temps	70%	91%	94%	79%	12%	79%
CC = 200000 juste à temps	64%	58%	70%	55%	21%	58%
CC = 2000 gestion des stocks	70%	70%	94%	42%	15%	33%
CC = 20000 gestion des stocks	64%	70%	94%	36%	18%	42%
CC = 200000 gestion des stocks	64%	48%	70%	33%	24%	45%

TAB. 8.3 – Comparaison du nombre de meilleures solutions obtenues sans l'option de partage de la demande selon les types d'instances : C, R et RC

8.1.2 Avec le GRASP

Nous utilisons pour tester le GRASP sous les différentes approches, la meilleure méthode en moyenne qui a été identifiée dans le chapitre 5.

Le tableau 8.4 indique le nombre de fois où les méthodes avec et sans partage de la demande trouvent les meilleurs résultats obtenus grâce au GRASP. Nous constatons, en juste à temps, que plus le coût de création d'une tournée augmente plus l'option de partager les demandes devient avantageuse. Au contraire, en gestion

Version	Nombre Min avec Partage	Nombre Min sans Partage
CC = 2000 juste à temps	36 %	64 %
CC = 20000 juste à temps	39 %	61 %
CC = 200000 juste à temps	47 %	53 %
CC = 2000 avec gestion des stocks	76 %	24 %
CC = 20000 avec gestion des stocks	32 %	68 %
CC = 200000 avec gestion des stocks	52 %	48 %

TAB. 8.4 – Comparaison des différentes versions selon les coûts de création d'une tournée avec le GRASP

des stocks, nous constatons que l'option partage de la demande est avantageuse mais l'écart entre le nombre de meilleures solutions obtenues par chaque méthode tend à se réduire avec l'augmentation des coûts.

	A	B	C	D	E	F	G	H	I	J	K
CC = 2000 juste à temps	53%	60%	64%	68%	65%	67%	65%	55%	71%	67%	69%
CC = 20000 juste à temps	52%	65%	52%	69%	59%	72%	59%	53%	63%	62%	65%
CC = 200000 juste à temps	46%	63%	10%	62%	40%	50%	60%	59%	65%	61%	65%
CC = 2000 gestion des stocks	20%	24%	24%	24%	28%	27%	20%	21%	27%	25%	29%
CC = 20000 gestion des stocks	68%	68%	61%	72%	69%	65%	71%	68%	68%	71%	71%
CC = 200000 gestion des stocks	48%	59%	12%	53%	37%	42%	60%	58%	54%	56%	47%

TAB. 8.5 – Comparaison des répartitions de meilleures solutions obtenues grâce au GRASP dans la version sans partage de la demande selon les catégories d'instances

Le tableau 8.5 indique le nombre de fois où la solution obtenue sans partage de la demande est meilleure que la solution obtenue avec partage de la demande en fonction des catégories des instances et des coûts de création imposés.

8.2 Pertinence de l'option gestion des stocks en fonction des coûts utilisés et de la physionomie de l'instance

	Nombre de meilleures solutions en juste à temps	Nombre de meilleures solutions avec gestion des stocks
CC 2000 CS 20 CR 50	39 %	61 %
CC 2000 CS 20 CR 100	40 %	60 %
CC 2000 CS 50 CR 20	42 %	59 %
CC 2000 CS100 CR 20	43 %	57 %

TAB. 8.6 – Comparaison du nombre de meilleures solutions obtenues selon l'option de gestion des stocks ou non et selon les coûts

Le tableau 8.6 permet d'étudier en fonction des coûts de stockage et de routage l'avantage ou non de gérer les stocks des magasins. Nous pouvons constater que parmi toutes les configurations de coûts que nous avons étudiées il est en moyenne plus intéressant de mettre en place une politique de gestion des stocks.

8.3 Les différents résultats obtenus sur les instances réelles

Les caractéristiques des instances réelles utilisées sont données dans le chapitre 3. Grâce aux méthodes heuristiques nous trouvons une solution pour IR1 de 77680000 et pour IR2 de 11414000. Le temps d'exécution pour trouver la solution heuristique à IR1 est de 927 s et celui pour obtenir la solution à IR2 est de 2176 s. Les tests prouvent donc que nos méthodes peuvent être appliquées à des réseaux de caractéristiques et de tailles différentes de ceux étudiés dans cette thèse.

8.4 Conclusion

Dans ce chapitre nous avons étudié les différents résultats obtenus sur le problème de tournées de véhicules dans la logistique inverse selon les différentes options mises en place. Tout d'abord nous avons réalisé une analyse de l'option qui autorise ou non à plusieurs véhicules de s'occuper de la même demande selon les coûts de création de tournées. Nous avons dans ce cas observé que cette option est d'autant plus intéressante que le coût de création d'une tournée est fort. Les instances les plus propices à la pratique d'une telle stratégie sont des instances ayant des petits sites. Nous avons également constaté que cette option était plus intéressante dans une politique avec gestion des stocks que sans. Nous montrons aussi que les méthodes heuristiques développées peuvent être mises en application sur les problèmes réels.

Chapitre 9

Conclusions et perspectives

Dans le cadre de cette thèse, nous avons étudié un problème de construction de tournées dans un contexte de logistique inverse. Plusieurs options de gestion et de distribution ont été traitées. Diverses méthodes de résolution ont été utilisées pour résoudre ce problème. Nous allons à présent résumer tour à tour les apports de cette thèse sur les deux objectifs fixés : les objectifs *"problème"* et les objectifs *"techniques"*. Nous terminerons ce chapitre en évoquant les différentes perspectives de travaux qui font suite à ceux exposés pendant cette thèse.

9.1 Conclusions

Dans cette thèse les objectifs étaient doubles. Tout d'abord les objectifs qualifiés de *problème* résidaient dans la mise en place d'un réseau générique, ceci dans le but de pouvoir être réutilisé dans divers contextes et de pouvoir étudier différentes politiques de gestion. Un éventail large d'options telles que la variation des différents coûts, la possibilité ou non de livrer un même client en plusieurs fois, la gestion ou non des stocks par le dépôt central a été proposée. Le problème a été traité dans un contexte de logistique inverse mais le réseau étudié s'adapte à d'autres problèmes dans un tout autre contexte. Nous avons résolu un problème original et modulable grâce à diverses méthodes. Nous avons pu détailler les différents résultats obtenus afin d'analyser la pertinence des différentes options selon les tailles et les physionomies des instances.

Quant aux objectifs *techniques*, ils résidaient dans l'emploi de techniques de programmation par contraintes au sein des méthodes de résolution via les méthodes hybrides. Nous avons dans un premier temps validé, grâce à des résultats intéressants, la cohérence de l'emploi de telles méthodes sur des problèmes comme celui que nous traitons. Puis dans un second temps nous avons comparé les performances des méthodes n'employant que des techniques issues de la recherche opérationnelle avec celles des méthodes hybrides. Trois types de méthodes ont été employés. Tout d'abord les méthodes heuristiques de type construction et améliorations ont été employées. Différentes techniques de constructions et d'amélioration ont tout d'abord été étudiées seules. La méthode de construction "meilleure insertion" (Solomon, 1987 [139] et Potvin et Rousseau, 1993 [124]) a été identifiée comme la plus adaptée à la résolution de notre problème. De même les techniques d'amélioration du *2-Opt*, *String Exchange* [155], *String Relocation* [155] et le déplacement de demande sur un jour antérieur ont montré leur efficacité lorsqu'elles ont été utilisées seules sur le problème. Finalement plusieurs combinaisons de méthodes d'amélioration faisant suite à une méthode de construction ont été testées. Nous avons trouvé qu'une méthode combinant meilleure insertion, *Or-Opt* [118], *2-échange*, *2-Opt*, *String Relocation* et *String Exchange* nommée $MI + E_i E_o D_o D_i O A^+$ était la plus adapté au traitement du problème en juste à temps par des méthodes heuristiques. De même une méthode combinant meilleure insertion, *2-Opt*, *String Relocation* et *String Exchange* nommée $MI + E_o D_o O^+$ a été identifiée comme la plus efficace pour résoudre le problème dans sa version avec gestion des stocks. Dans un second temps le problème traité a été résolu grâce à la métaheuristique GRASP. Tout d'abord utilisée avec des techniques de recherche locale issues de la recherche opérationnelle, nous avons par la suite hybridé la partie recherche locale de cette méthode en utilisant la technique LNS [137]. L'hybridation de la méthode GRASP n'a pas en moyenne donné de résultats meilleurs que ceux obtenus par des techniques dites classiques mais a permis d'améliorer certains résultats localement (sur certaines instances). Aucune configuration type des instances améliorées par l'hybridation n'a pu être mise en évidence mise à part son appartenance aux classes : R1 de la catégorie A, RC1 de la catégorie B et RC1 de la catégorie I. Le seul point commun à ses instances étant la présence de fenêtres de visite serrées et l'aspect aléatoire dans le positionnement des sites dans

l'espace. Finalement nous avons utilisé des techniques inspirées par les méthodes de résolution exacte par le biais d'une *génération de colonnes* [71] suivie d'un *Branch and Bound* [102]. La résolution du sous-problème dans le processus de génération de colonnes a tour à tour été réalisée grâce à de la programmation dynamique, une technique de recherche taboue suivie de la programmation dynamique et de la programmation par contraintes. Nous avons pu voir qu'une méthode de *Branch and Bound* donne des résultats concluants et suffisants pour les problèmes en juste à temps. Par contre pour le problème avec gestion des stocks une méthode de *Branch and Price* paraît plus appropriée. Les résultats prouvent la complémentarité des méthodes issues de la recherche opérationnelle et celles issues de la programmation par contraintes.

9.2 Perspectives

En ce qui concerne les différentes méthodes heuristiques, une première perspective serait l'emploi de techniques de programmation par contraintes au sein de la méthode de construction. Une fois toutes les demandes planifiées et donc attribuées à une tournée, nous chercherions pour chaque tournée quel est l'ordre de visite optimal minimisant le coût de transport tout en respectant les fenêtres de visite des sites et la capacité des véhicules. Une seconde perspective concerne une éventuelle utilisation industrielle. Il faudrait alors mutualiser l'ensemble des heuristiques testées dans ce chapitre afin de ne donner que la meilleure solution à un utilisateur qui ne chercherait pas le meilleur résultat en moyenne mais plutôt à toujours avoir la meilleure solution. En effet il est rare que toutes les heuristiques échouent sur la même instance, on a donc intérêt à exécuter toutes les heuristiques disponibles et à conserver la meilleure solution.

En ce qui concerne les perspectives sur la métaheuristique GRASP. Une première perspective de travail sur la méthode de résolution hybride serait de développer une contrainte globale permettant de gérer d'un seul bloc le problème de ré-optimisation lors de la recherche locale. Grâce à cette contrainte globale plus de filtrage pourrait être réalisé. Une seconde perspective serait d'utiliser la contrainte globale de chemin développée par Beldiceanu *et al.* [11]. En effet, dans leurs travaux les auteurs s'attachent à montrer comment prendre en compte de manière globale un certain nombre de restrictions (précédence entre sommets, incomparabilités entre sommets) au sein d'une contrainte de partitionnement de graphes par les arbres.

Les différentes perspectives concernant la résolution basée sur des méthodes exactes résident essentiellement dans le gain de temps permettant éventuellement la résolution d'instances plus grandes. Des techniques telles que l'utilisation de coupes, le calcul de bornes du problème, l'ajout d'inégalités valides, n'ont pas été suffisamment développées ici et pourraient permettre d'améliorer nos temps de résolution pour chacune des méthodes et plus particulièrement pour les techniques de programmation par contraintes. De même afin de gagner du temps nous pourrions mettre en place la technique LDS. Ainsi les gains de temps et de mémoire permettraient de faire évoluer les paramètres de tests concernant la division des demandes ou de pouvoir traiter des instances avec plus de sites. Une autre perspective serait le développement d'un *Branch and Price* en particulier pour la résolution du problème avec gestion des stocks.

Finalement des perspectives d'extension de la problématique sont envisagées telles qu'un réseau multi-dépôts ou encore un réseau prenant en compte plusieurs produits.

Quatrième partie

Annexes

Annexe 1 - Bases de la programmation par contraintes

Dans cette partie, nous allons aborder des notions et techniques de Programmation Par Contraintes qui nous sont utiles dans cette thèse. Cette partie ne se veut pas un cours exhaustif sur la programmation par contraintes.

Définition d'un problème de satisfaction de contraintes : CSP [1]

Pour définir un problème de satisfaction de contraintes, trois données sont essentielles. On se donne tout d'abord un ensemble $V = \{v_1, \ldots, v_n\}$ de n *variables*. À chaque variable v de V est associé un ensemble D_v appelé *domaine* de v. L'ensemble des domaines de définition associés à ces variables est nommé D, ainsi $D = \{D_1, ..., D_n\}$ où D_{v_1} représente le domaine de définition de la variable v_1 *etc.* Les domaines de définition des variables, peuvent être entiers, continus, ensemblistes *etc.* Enfin, il existe des relations liant les différentes variables entre elles et qui doivent être vérifiées par l'instanciation des différentes variables. Ces relations sont appelées *contraintes*, et l'ensemble des contraintes de notre problème sera nommé l'ensemble C. Ainsi, un problème de satisfaction de contraintes est défini par le triplet (V, D, C). On dit qu'une instanciation (même partielle) est consistante si elle satisfait l'ensemble des contraintes du problème. Inversement, une instanciation est dite inconsistante si elle ne satisfait pas l'ensemble des contraintes du problème.

Le CSP peut être binaire si les contraintes qui le composent ne portent que sur au plus deux variables. Cette définition pour être généralisée au rang n sous le nom CSP n-aires. Mais par binarisation, tous les CSP n-aires peuvent se ramener à un CSP binaire. On peut également trouver des CSP-booléen (où les variables ne peuvent prendre que la valeur *vrai* ou *faux*), ce genre de problème est appelé problème de satisfiabilité (appelé SAT).

Voyons dans les sous-sections suivantes les deux principes de résolution pour un CSP.

La réduction de l'espace de recherche - filtrage - propagation

Cette approche de résolution consiste en un raisonnement logique. Ainsi, grâce au croisement de plusieurs contraintes mettant en jeu des variables communes, des valeurs des domaines de ces variables peuvent être considérées comme impossibles et peuvent être enlevées des domaines de celles-ci. De cette façon, les domaines se retrouvent épurés des valeurs qui amènent à une solution non réalisable pour le problème (*c.-à-d.* où une ou plusieurs contraintes se retrouvent violées).

De même, on peut utiliser la notion de consistance d'arcs pour continuer à épurer certains domaines de variables. En effet, une contrainte liant deux variables X_i et X_j dans un CSP binaire est dite arc-consistante si pour toute valeur d_i de X_i, il existe une valeur d_j dans le domaine de X_j telle que l'instanciation partielle (d_i, d_j) satisfait la contrainte. Pour effectuer la consistance d'arc d'un CSP binaire citons les algorithmes : AC [2]-1, AC-3...On peut généraliser cette notion avec la k-consistance, ainsi un problème est dit k-consistant si toute solution partielle de $k-1$ variables peut être étendue à une solution partielle de k variables en choisissant n'importe quelle valeur dans le domaine de la k-ième variable.

Le CSP peut être résolu via ces méthodes que si l'on arrive à l'un des deux cas suivants :

– un des domaines des variables est vide, dans ce cas là aucune solution ne peut être trouvée à ce problème sans violer une ou plusieurs contraintes.

[1] Constraint Satisfaction Problem

[2] Arc Consistency

– le domaine de toutes les variables est réduit à un singleton et la consistance globale est vérifiée dans ce cas là une solution est trouvée.

L'énumération ou recherche en profondeur d'abord

La seconde façon de résoudre un CSP est l'énumération. L'énumération est dite implicite. Dans la recherche arborescente chaque nœud de l'arbre représente une instanciation partielle des différentes variables. Chaque nœud représente un point de choix de l'algorithme (affectation, découpage, *etc.*). Lorsqu'entre chaque nœud on réalise une affectation, la différence entre un nœud fils et son père est l'instanciation d'une nouvelle variable dans le nœud fils qui ne l'était pas dans le nœud père. Un nœud est ensuite étendu si l'ensemble des contraintes utilisant les variables instanciées sont satisfaites. En cas de non satisfaction d'une ou de plusieurs contraintes par la solution partielle *c.-à-d.* inconsistance de la solution partielle, un retour-arrière au nœud père est effectué. Il s'agit ici de la technique de *backtracking*. L'ordre de traitement des variables a donc un rôle important et plusieurs ont été étudiés (variable dont le domaine est le plus petit d'abord, la variable qui est la plus contrainte, *etc.*). De même, la valeur ou le sous ensemble de valeurs affectée à la variable considérée, peut être l'objet d'étude : ordre croissant dans le domaine, *etc.*

Couplage entre backtracking et propagation de contraintes

Afin d'améliorer la technique du backtracking, on peut coupler backtracking et propagation de contraintes. Ainsi, les problèmes menant à des insatisfaisabilités peuvent être décelés plus tôt. Citons forward-checking [86] qui réduit le domaine des variables, non encore instanciées, en fonction de la dernière instanciation de variable réalisée. L'arc-consistance peut aussi être maintenue par rapport à toutes les décisions prises auparavant comme dans real-full-look-ahead (rfl) [116] ou maintaining-arc-consistency (mac) [135].

Les contraintes globales

Une contrainte globale, est une contrainte qui porte sur un ensemble de variables et qui a pour équivalence un ensemble de contraintes élémentaires, mais avec laquelle on obtient de meilleurs résultats par le fait d'un niveau de consistance plus élevé. Ainsi les contraintes globales permettent de traiter en bloc certains sous-ensembles de contraintes pour améliorer la réduction des domaines et la recherche de solutions. Elles peuvent être vues comme une conjonction de contraintes ou une relaxation d'un problème plus grand. Un certain nombre de contraintes dites globales existent dans la littérature. Un des exemples classiques de contrainte globale est la contrainte *alldifferent()* [127]. Cette contrainte permet de donner une valeur différente à chacune des variables passées en paramètre de la contrainte. Par exemple si on a *alldifferent(x, y, z)* cela revient à avoir les contraintes suivantes : $x \neq y$, $x \neq z$ et $z \neq y$. Dans ce cas là, appliquer l'arc consistance sur la clique des contraintes binaires ne permet pas de supprimer en général autant de valeurs que si on l'applique directement sur la contrainte *alldifferent()*.

Annexe 2 - Bases de la programmation linéaire

Dans cette partie nous allons aborder des notions et techniques de Programmation Linéaire qui nous sont utiles dans cette thèse. Cette partie ne se veut pas un cours exhaustif sur la programmation linéaire.

Définition d'un Programme Linéaire

Un programme linéaire (PL) est un problème d'optimisation de la forme :

$$\begin{aligned} &\min \text{ ou } \max \quad z = \sum_{j=1}^{n} (c_j x_j) \\ &\forall\, i = \{1 \dots m\} \quad \sum_{j=1}^{n} \{a_{ij} x_j\} \leq \text{ ou } \geq \text{ ou } = \quad b_i \\ &\forall\, j = \{1 \dots n\} \quad x_j \quad \geq 0 \end{aligned}$$

Dans ce problème une fonction objectif z est à optimiser (minimisation ou maximisation). m relations entre les variables contraignent ce problème. Les variables sont au nombre de n et sont toutes positives ou nulles.

Définition du dual

Il est parfois plus intéressant de résoudre le dual d'un problème que sa formulation primal. En effet par exemple un problème qui ne comprend que deux contraintes mais plusieurs variables peut-être résolu graphiquement grâce à son dual puisqu'il n'y aura que deux variables. Ou encore un PL peut ne pas avoir de base réalisable évidente mais son dual oui il est donc plus facile de résoudre le dual que le primal.
Si nous prenons comme PL de départ (primal) :

$$\begin{aligned} &\max \quad z = \sum_{j=1}^{n} (c_j x_j) \\ &\forall\, i = \{1 \dots m\} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\ &\forall\, j = \{1 \dots n\} \quad x_j \quad \geq 0 \end{aligned}$$

En définissant les variables duales y_i. Nous pouvons alors écrire le dual de ce PL ainsi :

$$\begin{aligned} &\min \quad w = \sum_{i=1}^{m} (b_i y_i) \\ &\forall\, j = \{1 \dots n\} \quad \sum_{i=1}^{m} a_{ji} y_j \geq c_j \\ &\forall\, i = \{1 \dots m\} \quad y_i \geq 0 \end{aligned}$$

Certaines propriétés du dual sont intéressantes à noter ici :
- Le dual du dual est le primal ;
- Si les x_j et les y_i sont solutions respectivement du primal et du dual alors on a $z \leq w$;
- Le théorème de la dualité dit : si un primal et son dual ont chacun une solution réalisable, ils ont chacun une solution optimale et de plus ces solutions ont même valeur ;

– Le théorème des écarts complémentaires dit : deux solutions x_j et y_i du primal et du dual sont optimales si et seulement si :

1. $\forall\, i \quad y_i(b_i - \sum_{j=1}^{n} a_{ij}x_j) = 0$
2. $\forall\, j \quad x_j(c_j - \sum_{i=1}^{m} a_{ij}y_i) = 0$

Résolution d'un PL

Chaque contrainte définit un demi-espace. L'intersection de tous les demi-espaces formés par toutes les contraintes du problème, forme un polyèdre qui est un ensemble convexe. Ce polyèdre défini l'ensemble S qui représente l'ensemble des solutions réalisables du problème. La solution optimale de ce problème va se trouver à un point extrême de ce polyèdre.

Ainsi une première méthode de résolution d'un PL est le simplexe. Cette technique consiste à parcourir l'ensemble des sommets du polyèdre des solutions à la recherche de la solution optimale. Deux autres algorithmes peuvent être cités : la méthode de l'ellipsoïde [95] et les algorithmes projectifs [94].

Il est parfois plus facile de résoudre le dual d'un problème que le programme primal.

Les problèmes linéaires en nombres entiers (PLNE)

Dans certains problèmes les variables doivent prendre des valeurs entières pour être considérées comme valides. Dans ces cas là nous parlons de programme linéaire en nombres entiers. Voici la formulation de ce type de problème :

$$\begin{aligned} \min \quad & z = cx \\ & Ax \geq b \\ & x \in \mathbb{N}^n \end{aligned}$$

Résolution d'un PLNE

Une des méthodes pour résoudre exactement un PLNE est par séparation et évaluation (PSE ou Branch and Bound [102]). Cette méthode est une recherche arborescente "intelligente". En effet, l'arborescence des solutions est construite progressivement. À chaque nœud la solution partielle est évaluée pour savoir si elle a des chances de nous mener à la solution optimale. Ainsi on examine seulement les sommets qui semblent mener à une solution optimale.

Une évaluation par défaut peut être trouvée grâce à des méthodes comme la relaxation. Dans ce type de méthode les contraintes qui sont les plus difficiles à satisfaire sont relaxées. Ainsi, on obtient une solution pour un problème plus "facile" et cette solution constitue une borne inférieure du problème en entier. Nous pouvons citer comme technique de relaxation : la relaxation continue (on relaxe ici les contraintes d'intégrité du problème). Citons encore la relaxation Lagrangienne qui elle consiste à relâcher des contraintes que l'on juge difficiles en les dualisant (*c.-à-d.* lorsque ces contraintes là sont violées la fonction objectif se trouve pénalisée).

De même, lorsqu'un PLNE est écrit on peut identifier des groupes de contraintes. La méthode de décomposition permet d'identifier des sous-problèmes au problème général. Ainsi, la résolution du problème passe par les résolutions de tous les sous-problèmes qui le constituent. Citons ici la méthode de décomposition de Benders [14]. Cette méthode consiste à fixer un certain nombre de variables du problème, ainsi le problème se trouve simplifié.

On peut également coupler une PSE avec les générations de coupes. Dans ces cas là nous parlerons d'une méthode de Branch and Cut [81].

Une autre méthode pouvant être citée dans cette partie est la génération de colonnes [113]. En effet, cette méthode consiste à ne traiter qu'une partie des variables, à les fixer puis à partir de cette solution partielle, à ajouter de nouvelles variables non encore fixées et réitérer.

Annexe 3 - Détails des résultats des heuristiques de construction et améliorations

Résultats des tests sur les versions en juste à temps selon les catégories

Instance	Coût	Version
AFixe-C1	2297155,5	$PMI + E_iE_oD_oD_iO^+$
AFixe-C2	1674012,5	$MI + E_iE_oD_oD_iO^+$
AFixe-R1	3492725	$MI + E_iE_oD_oD_iO^+$
AFixe-R2	2549350	$MI + E_iE_oD_oD_iO^+$
AFixe-RC1	3099956,2	$MI + E_iE_oD_oD_iO^+$
AFixe-RC2	2440212,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
AGauss5-C1	2331906,6	$PMI + E_iE_oD_oD_iO^+$
AGauss5-C2	1676375	$MI + E_iE_oD_oD_iO^+$
AGauss5-R1	3425930,8	$MI + E_iE_oD_oD_iO^+$
AGauss5-R2	2543944,5	$MI + D_oD_iOE_iE_o$
AGauss5-RC1	3063377,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
AGauss5-RC2	2438308,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
AGauss15-C1	2332817,7	$PMI + E_iE_oD_oD_iO^+$
AGauss15-C2	1679851,2	$MI + E_iE_oD_oD_iO^+$
AGauss15-R1	3437458,3	$MI + E_iE_oD_oD_iO^+$
AGauss15-R2	2539019	$MI + E_iE_oD_oD_iO^+$
AGauss15-RC1	3090493,7	$MI + E_iE_oD_oD_iO^+$
AGauss15-RC2	2449966,2	$\overrightarrow{MI + E_iE_oD_oD_iO}$
BFixe-C1	2524455,5	$\overrightarrow{PMI + E_iE_oD_oD_iO}$
BFixe-C2	1712156,2	$MI + E_iE_oD_oD_iO^+$
BFixe-R1	3665362,5	$MI + E_iE_oD_oD_iO^+$
BFixe-R2	2546354,5	$MI + E_iE_oD_oD_iO^+$
BFixe-RC1	3114506,2	$MI + E_iE_oD_oD_iO^+$
BFixe-RC2	2452012,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
BGauss5-C1	2633843,3	$PMI + E_iE_oD_oD_iO^+$
BGauss5-C2	1704263,75	$MI + E_iE_oD_oD_iO^+$
BGauss5-R1	3645020	$MI + E_iE_oD_oD_iO^+$
BGauss5-R2	2557843,6	$MI + D_oD_iOE_iE_o$
BGauss5-RC1	3109638,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
BGauss5-RC2	2446578,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
BGauss15-C1	2586587,7	$MI + E_iE_oD_oD_iO^+$
BGauss15-C2	1751602,5	$MI + E_iE_oD_oD_iO^+$
BGauss15-R1	3656599,1	$MI + E_iE_oD_oD_iO^+$
BGauss15-R2	2549933,6	$MI + E_iE_oD_oD_iO^+$
BGauss15-RC1	3214457,5	$MI + E_iE_oD_oD_iO^+$
BGauss15-RC2	2445557,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
CFixe-C1	3542038,8	$MI + E_iE_oD_oD_iO^+$
CFixe-C2	2047137,5	$MI + E_iE_oD_oD_iO^+$
CFixe-R1	4568650	$MI + E_iE_oD_oD_iO^+$
CFixe-R2	2792690,9	$MI + E_iE_oD_oD_iO^+$
CFixe-RC1	5020706,2	$MI + E_iE_oD_oD_iO^+$
CFixe-RC2	2592281,2	$MI + E_iE_oD_oD_iO^+$
CGauss5-C1	3537508,8	$MI + E_iE_oD_oD_iO^+$
CGauss5-C2	2077418,7	$MI + E_iE_oD_oD_iO^+$
CGauss5-R1	4648760,8	$MI + E_iE_oD_oD_iO^+$
CGauss5-R2	2754221,8	$MI + E_iE_oD_oD_iO^+$

Instance	Coût	Version
CGauss5-RC1	5089225	$MI + E_iE_oD_oD_iO^+$
CGauss5-RC2	2576267,5	$MI + E_iE_oD_oD_iO^+$
CGauss15-C1	3538425,5	$MI + E_iE_oD_oD_iO^+$
CGauss15-C2	2060066,2	$MI + E_iE_oD_oD_iO^+$
CGauss15-R1	4654259,1	$MI + E_iE_oD_oD_iO^+$
CGauss15-R2	2701309	$MI + E_iE_oD_oD_iO^+$
CGauss15-RC1	5036846,2	$MI + E_iE_oD_oD_iO^+$
CGauss15-RC2	2581968,7	$MI + E_iE_oD_oD_iO^+$
DFixe-C1	2769955,5	$MI + E_iE_oD_oD_iO^+$
DFixe-C2	1817037,5	$MI + E_iE_oD_oD_iO^+$
DFixe-R1	3799662,5	$MI + E_iE_oD_oD_iO^+$
DFixe-R2	2596540,9	$MI + E_iE_oD_oD_iO^+$
DFixe-RC1	3852437,5	$MI + E_iE_oD_oD_iO^+$
DFixe-RC2	2448643,7	$MI + E_iE_oD_oD_iO^+$
DGauss5-C1	2691241,1	$MI + E_iE_oD_oD_iO^+$
DGauss5-C2	1813020	$MI + E_iE_oD_oD_iO^+$
DGauss5-R1	3755528,3	$MI + E_iE_oD_oD_iO^+$
DGauss5-R2	2598642,7	$MI + E_iE_oD_oD_iO^+$
DGauss5-RC1	3810453,7	$MI + E_iE_oD_oD_iO^+$
DGauss5-RC2	2449781,2	$\overrightarrow{MI + E_iE_oD_oD_iO}$
DGauss15-C1	2736353,3	$MI + E_iE_oD_oD_iO^+$
DGauss15-C2	1805622,5	$MI + E_iE_oD_oD_iO^+$
DGauss15-R1	3796168,3	$MI + E_iE_oD_oD_iO^+$
DGauss15-R2	2620155,4	$MI + E_iE_oD_oD_iO^+$
DGauss15-RC1	3813435	$MI + E_iE_oD_oD_iO^+$
DGauss15-RC2	2487543,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
EFixe-C1	3189144,4	$MI + E_iE_oD_oD_iO^+$
EFixe-C2	2007668,7	$MI + E_iE_oD_oD_iO^+$
EFixe-R1	4296937,5	$MI + E_iE_oD_oD_iO^+$
EFixe-R2	2642277,2	$MI + E_iE_oD_oD_iO^+$
EFixe-RC1	4407256,2	$\overrightarrow{MI + E_iE_oD_oD_iO}$
EFixe-RC2	2492600	$MI + E_iE_oD_oD_iO^+$
EGauss5-C1	3105630	$MI + E_iE_oD_oD_iO^+$
EGauss5-C2	2019447,5	$MI + E_iE_oD_oD_iO^+$
EGauss5-R1	4371700,8	$MI + E_iE_oD_oD_iO^+$
EGauss5-R2	2649715,4	$MI + E_iE_oD_oD_iO^+$
EGauss5-RC1	4440732,5	$MI + E_iE_oD_oD_iO^+$
EGauss5-RC2	2388922,5	$MI + E_iE_oD_oD_iO^+$
EGauss15-C1	3150837,7	$MI + E_iE_oD_oD_iO^+$
EGauss15-C2	2001758,7	$MI + E_iE_oD_oD_iO^+$
EGauss15-R1	4322540,8	$MI + E_iE_oD_oD_iO^+$
EGauss15-R2	2662680	$MI + E_iE_oD_oD_iO^+$
EGauss15-RC1	4388345	$MI + E_iE_oD_oD_iO^+$
EGauss15-RC2	2410581,2	$\overrightarrow{MI + E_iE_oD_oD_iO}$
FFixe-C1	2939011,1	$MI + E_iE_oD_oD_iO^+$
FFixe-C2	1941125	$\overrightarrow{MI + E_iE_oD_oD_iO}$
FFixe-R1	4052633,3	$MI + E_iE_oD_oD_iO^+$
FFixe-R2	22619122,7	$MI + E_iE_oD_oD_iO^+$
FFixe-RC1	4126468,7	$MI + E_iE_oD_oD_iO^+$
FFixe-RC2	2400631,25	$\overrightarrow{MI + E_iE_oD_oD_iO}$

Instance	Coût	Version
FGauss5-C1	2936694,4	$MI + E_iE_oD_oD_iO^+$
FGauss5-C2	1892692,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
FGauss5-R1	4061292,5	$MI + E_iE_oD_oD_iO^+$
FGauss5-R2	2611023,6	$MI + E_iE_oD_oD_iO^+$
FGauss5-RC1	4159913,7	$MI + E_iE_oD_oD_iO^+$
FGauss5-RC2	2412793,7	$MI + E_iE_oD_oD_iO^+$
FGauss15-C1	2955200	$MI + E_iE_oD_oD_iO^+$
FGauss15-C2	1894813,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
FGauss15-R1	4027925,8	$MI + E_iE_oD_oD_iO^+$
FGauss15-R2	2629105,4	$MI + E_iE_oD_oD_iO^+$
FGauss15-RC1	4167203,7	$MI + E_iE_oD_oD_iO^+$
FGauss15-RC2	2456020	$MI + E_iE_oD_oD_iO^+$
GFixe-C1	2487361,1	$PMI + E_iE_oD_oD_iO^+$
GFixe-C2	1752637,5	$MI + E_iE_oD_oD_iO^+$
GFixe-R1	3759308,3	$MI + E_iE_oD_oD_iO^+$
GFixe-R2	2550281,8	$MI + E_iE_oD_oD_iO^+$
GFixe-RC1	3547800	$\overrightarrow{MI + E_iE_oD_oD_iO}$
GFixe-RC2	2440618,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
GGauss5-C1	2538497,7	$MI + E_iE_oD_oD_iO^+$
GGauss5-C2	1760318,7	$MI + E_iE_oD_oD_iO^+$
GGauss5-R1	3721330	$MI + E_iE_oD_oD_iO^+$
GGauss5-R2	2550417,2	$MI + E_iE_oD_oD_iO^+$
GGauss5-RC1	3475838,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
GGauss5-RC2	2440618,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
GGauss15-C1	2551162,2	$MI + E_iE_oD_oD_iO^+$
GGauss15-C2	1745926,2	$MI + E_iE_oD_oD_iO^+$
GGauss15-R1	3719577,5	$MI + E_iE_oD_oD_iO^+$
GGauss15-R2	2575828,1	$MI + E_iE_oD_oD_iO^+$
GGauss15-RC1	3490585	$MI + E_iE_oD_oD_iO^+$
GGauss15-RC2	2445480	$\overrightarrow{MI + E_iE_oD_oD_iO}$
HFixe-C1	2326511,1	$MI + E_iE_oD_oD_iO^+$
HFixe-C2	1703787,5	$MI + E_iE_oD_oD_iO^+$
HFixe-R1	3480725	$MI + E_iE_oD_oD_iO^+$
HFixe-R2	2549768,1	$MI + E_iE_oD_oD_iO^+$
HFixe-RC1	3272062,5	$MI + E_iE_oD_oD_iO^+$
HFixe-RC2	2442187,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
HGauss5-C1	2390010	$MI + E_iE_oD_oD_iO^+$
HGauss5-C2	1735375	$MI + E_iE_oD_oD_iO^+$
HGauss5-R1	3463000,8	$MI + E_iE_oD_oD_iO^+$
HGauss5-R2	2549236,3	$MI + E_iE_oD_oD_iO^+$
HGauss5-RC1	3282430	$MI + E_iE_oD_oD_iO^+$
HGauss5-RC2	2439120	$\overrightarrow{MI + E_iE_oD_oD_iO}$
HGauss15-C1	2411236,6	$MI + E_iE_oD_oD_iO^+$
HGauss15-C2	1694370	$MI + E_iE_oD_oD_iO^+$
HGauss15-R1	3509369,1	$MI + E_iE_oD_oD_iO^+$
HGauss15-R2	2553719	$MI + D_oD_iOE_iE_o$
HGauss15-RC1	3268997,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
HGauss15-RC2	2428133,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
IFixe-C1	2473988,8	$MI + E_iE_oD_oD_iO^+$
IFixe-C2	1806900	$MI + E_iE_oD_oD_iO^+$
IFixe-R1	3797025	$MI + E_iE_oD_oD_iO^+$
IFixe-R2	2596540	$MI + E_iE_oD_oD_iO^+$

Instance	Coût	Version
IFixe-RC1	3630656,2	$MI + E_iE_oD_oD_iO^+$
IFixe-RC2	2452587,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
IGauss5-C1	2590782,2	$MI + E_iE_oD_oD_iO^+$
IGauss5-C2	1833233,7	$MI + E_iE_oD_oD_iO^+$
IGauss5-R1	3775895	$MI + E_iE_oD_oD_iO^+$
IGauss5-R2	2597909	$\overrightarrow{MI + E_iE_oD_oD_iO}$
IGauss5-RC1	3608535	$MI + E_iE_oD_oD_iO^+$
IGauss5-RC2	2448347,5	$\overrightarrow{MI + E_iE_oD_oD_iO}$
IGauss15-C1	2592232,2	$MI + E_iE_oD_oD_iO^+$
IGauss15-C2	1811463,7	$MI + E_iE_oD_oD_iO^+$
IGauss15-R1	3772546,6	$MI + E_iE_oD_oD_iO^+$
IGauss15-R2	2603492,7	$MI + E_iE_oD_oD_iO^+$
IGauss15-RC1	3598606,2	$MI + E_iE_oD_oD_iO^+$
IGauss15-RC2	2445491,2	$MI + E_iE_oD_oD_iO^+$
JFixe-C1	2434833,3	$MI + E_iE_oD_oD_iO^+$
JFixe-C2	1723825	$MI + E_iE_oD_oD_iO^+$
JFixe-R1	3553975	$MI + E_iE_oD_oD_iO^+$
JFixe-R2	2555504,5	$MI + E_iE_oD_oD_iO^+$
JFixe-RC1	3389306,2	$\overrightarrow{MI + E_iE_oD_oD_iO}$
JFixe-RC2	2450300	$\overrightarrow{MI + E_iE_oD_oD_iO}$
JGauss5-C1	2544920	$MI + E_iE_oD_oD_iO^+$
JGauss5-C2	1722193,7	$MI + E_iE_oD_oD_iO^+$
JGauss5-R1	3547250	$MI + E_iE_oD_oD_iO^+$
JGauss5-R2	2553044,5	$MI + D_oD_iOE_iE_o$
JGauss5-RC1	3369640	$\overrightarrow{MI + E_iE_oD_oD_iO}$
JGauss5-RC2	2452353,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$
JGauss15-C1	2568153,3	$MI + E_iE_oD_oD_iO^+$
JGauss15-C2	1728532,5	$MI + E_iE_oD_oD_iO^+$
JGauss15-R1	3597991,6	$MI + E_iE_oD_oD_iO^+$
JGauss15-R2	2564127,2	$MI + D_oD_iOE_iE_o$
JGauss15-RC1	3424116,2	$MI + E_iE_oD_oD_iO^+$
JGauss15-RC2	2455096,2	$\overrightarrow{MI + E_iE_oD_oD_iO}$
KFixe-C1	2407050	$MI + E_iE_oD_oD_iO^+$
KFixe-C2	1674012,5	$MI + E_iE_oD_oD_iO^+$
KFixe-R1	3492725	$MI + E_iE_oD_oD_iO^+$
KFixe-R2	2549350	$MI + E_iE_oD_oD_iO^+$
KFixe-RC1	3099956,2	$MI + E_iE_oD_oD_iO^+$
KFixe-RC2	2442437,5	$MI + E_iE_oD_oD_iO^+$
KGauss5-C1	2371568,8	$MI + E_iE_oD_oD_iO^+$
KGauss5-C2	1676375	$MI + E_iE_oD_oD_iO^+$
KGauss5-R1	3425930,8	$MI + E_iE_oD_oD_iO^+$
KGauss5-R2	2547871,8	$MI + E_iE_oD_oD_iO^+$
KGauss5-RC1	3063465	$MI + E_iE_oD_oD_iO^+$
KGauss5-RC2	2440086,2	$MI + E_iE_oD_oD_iO^+$
KGauss15-C1	2366615,5	$MI + E_iE_oD_oD_iO^+$
KGauss15-C2	1679851,2	$MI + E_iE_oD_oD_iO^+$
KGauss15-R1	3437458,3	$MI + E_iE_oD_oD_iO^+$
KGauss15-R2	2539019	$MI + E_iE_oD_oD_iO^+$
KGauss15-RC1	3090493,7	$MI + E_iE_oD_oD_iO^+$
KGauss15-RC2	2450078,7	$\overrightarrow{MI + E_iE_oD_oD_iO}$

TAB. 9.1 – Résultats obtenus dans les tests des versions en juste à temps

Résultats des tests sur les versions avec gestion des stocks selon les catégories

Instance	Coût	Version
AFixe-C1	2052253,3	$PMI + E_iE_oD_oD_iOA^+$
AFixe-C2	1629547,5	$MI + E_iE_oD_oD_iOA^+$
AFixe-R1	3394500,8	$MI + OE_iE_oAD_oD_i$
AFixe-R2	2515060,9	$MI + OE_iE_oAD_oD_i$
AFixe-RC1	3075633,7	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
AFixe-RC2	2431796,2	$MI + OE_iE_oAD_oD_i$
AGauss5-C1	2060490	$PMI + E_iE_oD_oD_iOA^+$
AGauss5-C2	1622448,7	$MI + E_iE_oD_oD_iOA^+$
AGauss5-R1	3330257,5	$MI + OE_iE_oAD_oD_i$
AGauss5-R2	2526674,5	$MI + OE_iE_oAD_oD_i$
AGauss5-RC1	3049332,5	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
AGauss5-RC2	2433466,2	$MI + OE_iE_oAD_oD_i$
AGauss15-C1	2007842,2	$PMI + E_iE_oD_oD_iOA^+$
AGauss15-C2	1629643,7	$MI + OE_iE_oAD_oD_i$
AGauss15-R1	3354065,8	$MI + OE_iE_oAD_oD_i$
AGauss15-R2	2525864,5	$MI + OE_iE_oAD_oD_i$
AGauss15-RC1	3058893,7	$MI + E_iE_oD_oD_iOA^+$
AGauss15-RC2	2447031,2	$MI + E_iE_oD_oD_iOA^+$
BFixe-C1	2447332,2	$MI + E_iE_oD_oD_iOA^+$
BFixe-C2	1670196,2	$MI + E_iE_oD_oD_iOA^+$
BFixe-R1	3582611,6	$MI + E_iE_oD_oD_iOA^+$
BFixe-R2	2531261,8	$MI + OE_iE_oAD_oD_i$
BFixe-RC1	3071146,2	$MI + E_iE_oD_oD_iOA^+$
BFixe-RC2	2445077,5	$MI + E_iE_oD_oD_iOA^+$
BGauss5-C1	2429080	$MI + E_iE_oD_oD_iOA^+$
BGauss5-C2	1660630	$MI + E_iE_oD_oD_iOA^+$
BGauss5-R1	3555695	$MI + E_iE_oD_oD_iOA^+$
BGauss5-R2	2535642,7	$MI + OE_iE_oAD_oD_i$
BGauss5-RC1	3071750	$MI + E_iE_oD_oD_iOA^+$
BGauss5-RC2	2444078,7	$MI + E_iE_oD_oD_iOA^+$
BGauss15-C1	2386112,2	$MI + E_iE_oD_oD_iOA^+$
BGauss15-C2	1705165	$MI + E_iE_oD_oD_iOA^+$
BGauss15-R1	3545897,5	$MI + E_iE_oD_oD_iOA^+$
BGauss15-R2	2522665,4	$MI + OE_iE_oAD_oD_i$
BGauss15-RC1	3144700	$MI + E_iE_oD_oD_iOA^+$
BGauss15-RC2	2447511,2	$MI + E_iE_oD_oD_iOA^+$
CFixe-C1	3509648,8	$MI + E_iE_oD_oD_iOA^+$
CFixe-C2	2027457,5	$MI + E_iE_oD_oD_iOA^+$
CFixe-R1	4437841,6	$MI + E_iE_oD_oD_iOA^+$
CFixe-R2	2729747,2	$MI + E_iE_oD_oD_iOA^+$
CFixe-RC1	4869307,5	$MI + E_iE_oD_oD_iOA^+$
CFixe-RC2	2495738,7	$MI + E_iE_oD_oD_iOA^+$
CGauss5-C1	3433594,4	$MI + E_iE_oD_oD_iOA^+$
CGauss5-C2	2035371,2	$MI + E_iE_oD_oD_iOA^+$
CGauss5-R1	4487543,3	$MI + E_iE_oD_oD_iOA^+$
CGauss5-R2	2682502,7	$MI + E_iE_oD_oD_iOA^+$
CGauss5-RC1	4942547,5	$MI + E_iE_oD_oD_iOA^+$
CGauss5-RC2	2482900	$MI + E_iE_oD_oD_iOA^+$
CGauss15-C1	3470864,4	$MI + E_iE_oD_oD_iOA^+$
CGauss15-C2	1996898,7	$MI + E_iE_oD_oD_iOA^+$

Instance	Coût	Version
CGauss15-R1	4465035,8	$MI + E_iE_oD_oD_iOA^+$
CGauss15-R2	2628171,8	$MI + E_iE_oD_oD_iOA^+$
CGauss15-RC1	4890987,5	$MI + E_iE_oD_oD_iOA^+$
CGauss15-RC2	2502258,7	$MI + E_iE_oD_oD_iOA^+$
DFixe-C1	2566412,2	$MI + E_iE_oD_oD_iOA^+$
DFixe-C2	1753585	$MI + E_iE_oD_oD_iOA^+$
DFixe-R1	3686020,8	$MI + E_iE_oD_oD_iOA^+$
DFixe-R2	2580620	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
DFixe-RC1	3712315	$MI + E_iE_oD_oD_iOA^+$
DFixe-RC2	2445497,5	$MI + E_iE_oD_oD_iOA^+$
DGauss5-C1	2516294,4	$MI + E_iE_oD_oD_iOA^+$
DGauss5-C2	11761376,2	$MI + E_iE_oD_oD_iOA^+$
DGauss5-R1	3656758,3	$MI + E_iE_oD_oD_iOA^+$
DGauss5-R2	2564925,4	$MI + OE_iE_oAD_oD_i$
DGauss5-RC1	3690495	$MI + E_iE_oD_oD_iOA^+$
DGauss5-RC2	2449332,5	$MI + E_iE_oD_oD_iOA^+$
DGauss15-C1	2515002,2	$MI + E_iE_oD_oD_iOA^+$
DGauss15-C2	1754525	$MI + E_iE_oD_oD_iOA^+$
DGauss15-R1	3671831,6	$MI + E_iE_oD_oD_iOA^+$
DGauss15-R2	2587512,7	$MI + E_iE_oD_oD_iOA^+$
DGauss15-RC1	3705247,5	$MI + E_iE_oD_oD_iOA^+$
DGauss15-RC2	2481522,5	$MI + OE_iE_oAD_oD_i$
EFixe-C1	3129701,1	$MI + E_iE_oD_oD_iOA^+$
EFixe-C2	1946081,2	$MI + E_iE_oD_oD_iOA^+$
EFixe-R1	4113185	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EFixe-R2	2559080	$MI + E_iE_oD_oD_iOA^+$
EFixe-RC1	4250795	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EFixe-RC2	2411248,7	$MI + E_iE_oD_oD_iOA^+$
EGauss5-C1	3049751,1	$MI + E_iE_oD_oD_iOA^+$
EGauss5-C2	1905410	$MI + E_iE_oD_oD_iOA^+$
EGauss5-R1	4098720,8	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EGauss5-R2	2592473,6	$MI + E_iE_oD_oD_iOA^+$
EGauss5-RC1	4278988,7	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EGauss5-RC2	2345308,7	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EGauss15-C1	3083277,7	$MI + E_iE_oD_oD_iOA^+$
EGauss15-C2	1876777,5	$MI + E_iE_oD_oD_iOA^+$
EGauss15-R1	4088660,8	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EGauss15-R2	2591232,7	$MI + E_iE_oD_oD_iOA^+$
EGauss15-RC1	4260476,2	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
EGauss15-RC2	2357571,2	$MI + E_iE_oD_oD_iOA^+$
FFixe-C1	2917510	$MI + E_iE_oD_oD_iOA^+$
FFixe-C2	1824785	$MI + E_iE_oD_oD_iOA^+$
FFixe-R1	3957100,8	$MI + E_iE_oD_oD_iOA^+$
FFixe-R2	2567769	$MI + E_iE_oD_oD_iOA^+$
FFixe-RC1	4016113,7	$MI + E_iE_oD_oD_iOA^+$
FFixe-RC2	2402666,2	$MI + E_iE_oD_oD_iOA^+$
FGauss5-C1	2857387,7	$MI + E_iE_oD_oD_iOA^+$
FGauss5-C2	1826210	$MI + E_iE_oD_oD_iOA^+$
FGauss5-R1	33968405	$MI + E_iE_oD_oD_iOA^+$
FGauss5-R2	2582682,7	$MI + E_iE_oD_oD_iOA^+$
FGauss5-RC1	4022972,5	$MI + E_iE_oD_oD_iOA^+$
FGauss5-RC2	2411570	$MI + E_iE_oD_oD_iOA^+$
FGauss15-C1	2831567,7	$MI + E_iE_oD_oD_iOA^+$

Instance	Coût	Version
FGauss15-C2	1810535	$MI + E_iE_oD_oD_iOA^+$
FGauss15-R1	3906220,8	$MI + E_iE_oD_oD_iOA^+$
FGauss15-R2	2599703,6	$MI + E_iE_oD_oD_iOA^+$
FGauss15-RC1	4070402,5	$MI + E_iE_oD_oD_iOA^+$
FGauss15-RC2	2453657,5	$MI + E_iE_oD_oD_iOA^+$
GFixe-C1	2373346,6	$MI + E_iE_oD_oD_iOA^+$
GFixe-C2	1737878,7	$MI + E_iE_oD_oD_iOA^+$
GFixe-R1	3640700,8	$MI + E_iE_oD_oD_iOA^+$
GFixe-R2	2518501,8	$MI + OE_iE_oAD_oD_i$
GFixe-RC1	3437832,5	$MI + E_iE_oD_oD_iOA^+$
GFixe-RC2	2446273,7	$MI + E_iE_oD_oD_iOA^+$
GGauss5-C1	2327245,5	$MI + E_iE_oD_oD_iOA^+$
GGauss5-C2	1719700	$MI + E_iE_oD_oD_iOA^+$
GGauss5-R1	3630685	$MI + E_iE_oD_oD_iOA^+$
GGauss5-R2	2525595,4	$MI + OE_iE_oAD_oD_i$
GGauss5-RC1	3442775	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
GGauss5-RC2	2446366,2	$MI + E_iE_oD_oD_iOA^+$
GGauss15-C1	2357995,5	$PMI + E_iE_oD_oD_iOA^+$
GGauss15-C2	1695561,2	$MI + E_iE_oD_oD_iOA^+$
GGauss15-R1	3629943,3	$MI + E_iE_oD_oD_iOA^+$
GGauss15-R2	2535591,8	$MI + OE_iE_oAD_oD_i$
GGauss15-RC1	3438222,5	$MI + E_iE_oD_oD_iOA^+$
GGauss15-RC2	2450596,2	$MI + E_iE_oD_oD_iOA^+$
HFixe-C1	2083718,8	$MI + E_iE_oD_oD_iOA^+$
HFixe-C2	1641035	$MI + OE_iE_oAD_oD_i$
HFixe-R1	3409840,8	$MI + E_iE_oD_oD_iOA^+$
HFixe-R2	2520220	$MI + OE_iE_oAD_oD_i$
HFixe-RC1	33181786,2	$MI + OE_iE_oAD_oD_i$
HFixe-RC2	2435948,7	$MI + E_iE_oD_oD_iOA^+$
HGauss5-C1	2136541,1	$MI + E_iE_oD_oD_iOA^+$
HGauss5-C2	1643011,2	$MI + OE_iE_oAD_oD_i$
HGauss5-R1	3406328,3	$MI + E_iE_oD_oD_iOA^+$
HGauss5-R2	2532959	$MI + OE_iE_oAD_oD_i$
HGauss5-RC1	3201545	$MI + OE_iE_oAD_oD_i$
HGauss5-RC2	2435791,2	$MI + E_iE_oD_oD_iOA^+$
HGauss15-C1	2134192,2	$MI + E_iE_oD_oD_iOA^+$
HGauss15-C2	1656857,5	$MI + E_iE_oD_oD_iOA^+$
HGauss15-R1	3432392,5	$MI + E_iE_oD_oD_iOA^+$
HGauss15-R2	2537201,8	$MI + OE_iE_oAD_oD_i$
HGauss15-RC1	3235468,7	$MI + E_iE_oD_oD_iOA^+$
HGauss15-RC2	2426673,7	$MI + E_iE_oD_oD_iOA^+$
IFixe-C1	2407172,2	$MI + E_iE_oD_oD_iOA^+$
IFixe-C2	1721592,5	$MI + E_iE_oD_oD_iOA^+$
IFixe-R1	3702116,6	$MI + E_iE_oD_oD_iOA^+$
IFixe-R2	2553132,7	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
IFixe-RC1	3571862,5	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
IFixe-RC2	2453027,5	$MI + E_iE_oD_oD_iOA^+$
IGauss5-C1	2433891,1	$MI + E_iE_oD_oD_iOA^+$
IGauss5-C2	1765471,2	$MI + E_iE_oD_oD_iOA^+$
IGauss5-R1	3694948,3	$MI + E_iE_oD_oD_iOA^+$

Instance	Coût	Version
IGauss5-R2	2553090	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
IGauss5-RC1	3533170	$\overrightarrow{MI + E_iE_oD_oD_iOA}$
IGauss5-RC2	2447425	$MI + E_iE_oD_oD_iOA^+$
IGauss15-C1	2432761,1	$MI + E_iE_oD_oD_iOA^+$
IGauss15-C2	1730552,5	$MI + E_iE_oD_oD_iOA^+$
IGauss15-R1	3663090,8	$MI + E_iE_oD_oD_iOA^+$
IGauss15-R2	2567374,5	$MI + OE_iE_oAD_oD_i$
IGauss15-RC1	3524248,7	$MI + E_iE_oD_oD_iOA^+$
IGauss15-RC2	2441927,5	$MI + E_iE_oD_oD_iOA^+$
JFixe-C1	2248143,3	$MI + E_iE_oD_oD_iOA^+$
JFixe-C2	1676062,5	$MI + E_iE_oD_oD_iOA^+$
JFixe-R1	3503453,3	$MI + E_iE_oD_oD_iOA^+$
JFixe-R2	2541670,9	$MI + OE_iE_oAD_oD_i$
JFixe-RC1	3218645	$MI + E_iE_oD_oD_iOA^+$
JFixe-RC2	2444186,2	$MI + E_iE_oD_oD_iOA^+$
JGauss5-C1	2352868,8	$MI + E_iE_oD_oD_iOA^+$
JGauss5-C2	1694663,7	$MI + E_iE_oD_oD_iOA^+$
JGauss5-R1	3487937,5	$MI + E_iE_oD_oD_iOA^+$
JGauss5-R2	2544080,9	$MI + OE_iE_oAD_oD_i$
JGauss5-RC1	3277072,5	$MI + E_iE_oD_oD_iOA^+$
JGauss5-RC2	2453035	$MI + E_iE_oD_oD_iOA^+$
JGauss15-C1	2372367,7	$MI + E_iE_oD_oD_iOA^+$
JGauss15-C2	1692207,5	$MI + E_iE_oD_oD_iOA^+$
JGauss15-R1	3514498,3	$MI + E_iE_oD_oD_iOA^+$
JGauss15-R2	2546900	$MI + OE_iE_oAD_oD_i$
JGauss15-RC1	3309285	$MI + E_iE_oD_oD_iOA^+$
JGauss15-RC2	2452438,7	$MI + E_iE_oD_oD_iOA^+$
KFixe-C1	2458334,4	$MI + E_iE_oD_oD_iOA^+$
KFixe-C2	1761985	$MI + E_iE_oD_oD_iOA^+$
KFixe-R1	3754578,3	$MI + E_iE_oD_oD_iOA^+$
KFixe-R2	2555399	$MI + OE_iE_oAD_oD_i$
KFixe-RC1	3520481,2	$MI + E_iE_oD_oD_iOA^+$
KFixe-RC2	2458128,7	$MI + E_iE_oD_oD_iOA^+$
KGauss5-C1	2427122,2	$MI + E_iE_oD_oD_iOA^+$
KGauss5-C2	1777306,2	$MI + E_iE_oD_oD_iOA^+$
KGauss5-R1	3739611,6	$MI + E_iE_oD_oD_iOA^+$
KGauss5-R2	2584662,7	$MI + E_iE_oD_oD_iOA^+$
KGauss5-RC1	3511131,2	$MI + E_iE_oD_oD_iOA^+$
KGauss5-RC2	2449420	$MI + E_iE_oD_oD_iOA^+$
KGauss15-C1	2457081,1	$MI + E_iE_oD_oD_iOA^+$
KGauss15-C2	1760016,2	$MI + E_iE_oD_oD_iOA^+$
KGauss15-R1	3757219,1	$MI + E_iE_oD_oD_iOA^+$
KGauss15-R2	2580554,5	$MI + E_iE_oD_oD_iOA^+$
KGauss15-RC1	3498973,7	$MI + E_iE_oD_oD_iOA^+$
KGauss15-RC2	2446462,5	$MI + E_iE_oD_oD_iOA^+$

TAB. 9.2 – Résultats obtenus grâce aux tests avec gestion des stocks

Bibliographie

[1] S. Anily. The general EOQ problem with vehicle routing costs. *European Journal of Operations Research*, 79 :451–473, 1994.

[2] S. Anily and A. Federgruen. Two-echelon distribution systems with vehicle routing costs and central inventories. *Operations Research*, 41(1) :37–47, 1993.

[3] S. Areibi and A. Vannelli. A GRASP clustering technique for circuit partitioning. In J. Gu and P.M. Pardalos, editors, *Satisfiability problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 711–724. American Mathematical Society, 1997.

[4] R. Baldacci, P. Toth, and D. Vigo. Recent advances in vehicle routing exact algorithms. *4OR : A Quarterly Journal of Operations Research*, 5(4) :269–298, 2007.

[5] S. Baptista, R.C. Oliveira, and E. Zúquete. A period vehicle routing case study. *European Journal of Operational Researc*, 139(2) :220–229, 2002.

[6] F. Bard, L. Huang, M. Dror, and P. Jaillet. A branch and cut algorithm for the vrp with satellite facilities. *IIE Transactions*, 30 :821 – 834, 1998.

[7] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch and price : Column generation for solving huge integer programs. *Operations Research*, 46(3) :316–329, 1998.

[8] A.I. Barros, R. Dekker, and V. Scholten. A two-level network for recycling sand : A case study. *European Journal of Operational Research*, 110(2) :199–214, 1998.

[9] J. Bautista and J. Pereira. Procedimientos para la localización de áreas de aportación de residuos urbanos. In *27 Congreso Nacional de Estadística e Investigación Operativa*, Lleida, Spain, April 2003.

[10] M. Beaulieu, R. Martin, and S. Landry. Logistique à rebours : un portrait nord-américain. *Logistics & Management*, 7 :5–14, 1999.

[11] N. Beldiceanu, P. Flener, and X. Lorca. Combining tree partitioning, precedence, incomparability, and degree constraints, with an application to phylogenetic and ordered-path problems. Technical Report Technical Report 2006-020, Department of Information Technology, Uppsala University,Sweden, 2006.

[12] W.J. Bell, L.M. Dalberto, M.L. Fisher, A.J. GreenField, R. Jaikumar, P. Kedia, R.G. Mack, and P.J. Prutzman. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces*, 13 :4–23, 1983.

[13] J.L. Beltran and D. Krass. Dynamic lot sizing with returning items and disposals. *IIE Transactions*, 34 :437–448, 2002.

[14] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik 4*, pages 238–252, 1962.

[15] J. Berger and M. Barkaoui. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers & OR*, 31(12) :2037–2053, 2004.

[16] L. Bertazzi, G. Paletta, and M.G. Speranza. Inventory control on sequences of links with given transportation frequencies. *International Journal of Production Economics*, 59(1-3) :261–270, 1999.

[17] N. Bianchessi and G. Righini. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers and Operations Research*, 34(2) :578–594, 2007.

[18] J. M. Blœmhof-Ruwaard, M. S., and L. N. Van Wassenhove. The capacitated distribution and waste disposal problem. *European Journal of Operational Research*, 88 :490–503, 1994.

[19] D. Bommisetty, M. Dessouky, and L. Jacob. Scheduling collection of recyclable material at northern illinois university campus using a two-phase algorithm. *Computers & Industrial Engineering*, 35 :435–438, 1998.

[20] N. Bostel, P. Dejax, and Z. Lu. The design, planning and optimization of reverse logistics systems : a review. In A. Langevin and D. Riopel, editors, *Logistics systems : Design and Optimization*. Kluwer Academic Publishers, 2005.

[21] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i : Route construction and local search algorithms. *Transportation Science*, 39(1) :104–118, 2005.

[22] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part ii : Metaheuristics. *Transportation Science*, 39(1) :119–139, 2005.

[23] L.D. Burns, R.W. Hall, D.E. Blumenfeld, and C.F. Daganzo. Distribution strategies that minimize transportation and inventory costs. *Operations Research*, 33(3) :469–490, 1985.

[24] A. Campbell, LW. Clarke, and M. Savelsbergh. Inventory Routing in Practice. In *The Vehicle Routing Problem*, pages 109–128. SIAM, Philadelphia, PA, USA, 2002.

[25] B. Canel-Depitre. La « logistique inversée » : Réponse efficace au consommateur et au citoyen. In *4ème congrès « Les tendances du marketing en Europe »*, 2004.

[26] Y. Caseau and F. Laburthe. Heuristics for large constrained vehicle routing problems. *Journal of Heuristics*, 5(3) :281 – 303, 1999.

[27] L.M. Chan, A. Federgruen, and D. Simchi-Levi. Probabilistic analysis and pratical algorithms for inventory-routing models. *Operations Research*, 46(1) :96–106, 1998.

[28] I.-M. Chao, B.L. Golden, and E. Wasil. A new heuristic for the multi-depot vehicle routing problem that improves upon best-known solutions. *American Journal of Mathematical and Management Sciences*, 13(3) :371–406, 1993.

[29] M. Chouinard. *Système organisationnel et architecture d'un support d'information pour l'intégration des activités de logistique inversée au sein d'un centre de réadaptation*. PhD thesis, Université Laval, 2003.

[30] M. Christiansen. Decomposition of a Combined Inventory and Time Constrained Ship Routing Problem. *Transportation Science*, 33(1) :3–16, 1999.

[31] M. Christiansen and B. Nygreen. A method for solving ship routing problems with inventory constraints. *Annals of Operations Research*, 81 :357–378, 1998.

[32] N. Christofides and J.E. Beasley. The period routing problem. *Networks*, 14 :237–256, 1984.

[33] O. Cognasse. Reverse logistics : un engouement croissant. *Stratégie Logistique*, 74 :49 – 55, 2005.

[34] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, MM. Solomon, and F. Soumis. The vrp with time windows. In P. Toth and eds. D. Vigo, editors, *The Vehicle Routing Problem*, page 157–193. SIAM, Philadelphia, PA, USA, 2002.

[35] J.-F. Cordeau, G.Laporte, M.W.P. Savelsbergh, and D. Vigo. Transportation handbooks in operations research and management science. In Laporte G. eds. Barnhart C., editor, *Vehicle Routing*, volume 14, page 367–428. North-Holland, Amsterdam, 2007.

[36] J.F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. Technical Report CRT-2003-24, Centre for research on transportation, 2003.

[37] J.F. Cordeau and G. Laporte. A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR*, 39 :292–298, 2001.

[38] J.F. Cordeau and G. Laporte. The dial-a-ride problem : Variants, modeling issues and algorithms. *4OR - Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1 :89–101, 2003.

[39] T.G. Crainic, M. Gendreau, and P. Dejax. Dynamic and stochastic models for the allocation of empty containers. *Operations Research*, 41(1) :102–126, 1993.

[40] G.B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410, 1954.

[41] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8 :101–111, 1960.

[42] Br. De Backer, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuristics. *J. Heuristics*, 6(4) :501–523, 2000.

[43] E. Del Castillo and J.K. Cochran. Optimal short horizon distribution operations in reusable containers. *Journal of Operational Research Society*, 47(3 part 1 of 2) :48–60, 1996.

[44] X. Delorme, X. Gandibleux, and J. Rodriguez. Application de la métaheuristique grasp à la résolution d'un problème de capacité d'infrastructure ferroviaire. In *3ème journées francophones de recherche opérationnelle (Francoro III)*, 2001.

[45] G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis. VRP with pickup and delivery. In P. Toth and eds. D. Vigo, editors, *The Vehicle Routing Problem*, page 157–193. SIAM, Philadelphia, PA, USA, 2002.

[46] G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column Generation*. Springer, 2005.

[47] M. Desrochers. An algorithm for the shortest path problem with resource constraints. Technical Report G-88-27, GERAD, 1988.

[48] M. Desrochers, J Desrosiers, and M.M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40 :342–354, 1992.

[49] M. Desrochers, J.K. Lenstra, and M.W.P. Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46 :322–332, 1990.

[50] J. Desrosier, Y. Dumas, and F. Soumis. A dynamic programming solutions of the large-scale single-vehicule dial-a-ride problem with time windows. *American Journal of Mathematics and Management Science*, 3(6) :301 – 325, 1986.

[51] I. Dobos. Optimal production-inventory strategies for HMMS-type reverse logistics system. *Int. J. Production Economics*, 81-82 :351–360, 2003.

[52] E. Domenjoud, C. Kirchner, and J. Zhou. Technical report on transport scheduling system ROUTER. Technical report, septembre 1998.

[53] M. Dorigo and T. Stützle. Ant colony optimization. *MIT Press*, 2004.

[54] M. Dror and M. Ball. Inventory/routing : reduction from an annual to a short period problem. *Naval Research Logistics*, 34 :891–905, 1987.

[55] M. Dror and P. Trudeau. Split delivery routing. *Naval Research Logistic*, (37) :383 – 402, 1990.

[56] R. Duhaime, D. Riopel, and A. Langevin. Value analysis and optimization of reusable containers at canada post. *Interfaces*, 31(3 part 1 of 2) :3–15, 2001.

[57] Y. Dumas, J. Desrosiers, and F. Soumis. Large scale multi-vehicle dial-a-ride systems. Technical Report G-89-30, Ecole des Hautes Etudes Commerciales (Ecole des HEC), Montreal Canada, 1989.

[58] A. Federgruen and P. Zipkin. A combined vehicle routing and inventory allocation problem. *Operations Research*, 32 :1019–1037, 1984.

[59] D. Feillet, P. Dejax, and M. Gendreau. Planification tactique du transport de marchandises inter-usines : application au secteur automobile. *Journal Européen des Sysèmes Automatisés*, 36(1), 2002.

[60] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems. *Networks*, 44 :216–229, 2004.

[61] T.A. Feo and M. Resende. A probabilistic heuristic for a computanionally difficult set covering problem. *Operations Research Letters*, (8) :67–71, 1989.

[62] T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, 23 :881–895, 1996.

[63] P. Festa and G. Resende. An annoted bibliography of grasp. Technical Report TD-5WYSEW, AT & T Labs Research, 2004.

[64] M.L Fisher and R. Jaikumar. A decomposition algorithm for large-scale vehicle routing problems with branch and price. Technical Report Technical Report 78-11-05, Departement of Decision Sciences, University of Pennsylvania, 1978.

[65] M.L Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11 :109––124, 1981.

[66] M. Fleischmann. Quantitative models for reverse logistics. In *Lecture Notes in Economics and Mathematical Systems*, volume 501. Springer-Verlag, 2001.

[67] M. Fleischmann, J. M. Bloemhof-Ruwaard, R. Dekker, E. Van Der Laan, and J.A.E.E. Van Wassenhove. Invited review, quantitative models for reverse logisitcs : a review. *European Journal of Operational Research*, 103 :1–17, 1997.

[68] M. Fleischmann, R. Kuik, and R. Dekker. Controlling inventories with stochastic item returns : a basic model. *European Journal of Operational Research*, 138 :63–75, 2002.

[69] M. Gaudioso and G. Paletta. A heuristic for the periodic vehicle routing problem. *Transportation Science*, 26 :86–92, 1992.

[70] M. Gendreau, P. Dejax, D. Feillet, and C. Gueguen. Vehicle routing with time windows and split deliveries. Technical Report rapport interne 2006-851, 2006.

[71] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem : part I. *Operations Research*, 9 :849–859, 1961.

[72] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem : part II. *Operations Research*, 11 :863–888, 1963.

[73] F. Glover. Tabu search — part I. *ORSA Journal on Computing*, 1 :190—206, 1989.

[74] F. Glover. Tabu search — part II. *ORSA Journal on Computing*, 2 :4—32, 1990.

[75] E. Grellier, P. Dejax, and N. Jussien. Heuristiques de construction et améliorations pour les problèmes de tournées de livraisons multi-périodiques incluant les concepts de logistique inverse. Technical Report 07/1/AUTO, École Des Mines de Nantes, 2007.

[76] E. Grellier, P. Dejax, and N. Jussien. An inventory pick-up and delivery problem in the reverse logistics context : Optimization using a grasp and hybrid approach. In *7th Metaheuristics International Conference (MIC'2007)*, Montreal, Canada, June 2007.

[77] E. Grellier, P. Dejax, and N. Jussien. Problème de tournées de collectes et livraisons multi-périodique : résolution grâce au GRASP. In *5èmes journées Francophones de Recherche Opérationnelle (FRANCORO V) 8ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2007)*, Grenoble, France, February 2007.

[78] E. Grellier, P. Dejax, N. Jussien, and Z. Lu. A column generation model and constraint programming techniques for solving an inventory routing problem in mixed flows. In *Third international workshop on freight transportation and logistics (Odysseus 2006)*, Altea, Spain, May 2006.

[79] E. Grellier, P. Dejax, N. Jussien, and Z. Lu. Vehicle routing problem in mixed flows for reverse logistics : a modeling framework. In *International Conference on Information Systems, Logistics, and Supply Chain (ILS 2006)*, Lyon, France, May 2006.

[80] E. Grellier, P. Dejax, and Y. Mati. Modélisation et optimisation d'un problème de tournées de véhicules avec gestion de stocks. Research Report 04-6-AUTO, École des Mines de Nantes, Nantes, France, 2004.

[81] M. Grötschel and O. Holland. Solution of large-scale travelling salesman problems. *Mathematical Programming*, 51(2) :141– 202, 1991.

[82] S.M. Gupta and K. Taleb. Scheduling disassembly. *International Journal of Production Research*, 32(8) :1857–1866, 1994.

[83] K. Halse. *Modeling and Solving Complex Vehicle Routing Problems*. PhD thesis, Technical University of Denmark, DK-2800 Lyngby, Denmark, 1992.

[84] Ø. Halskau, I. Griblovskaia, and K.N.B. Myklebost. Models for pick-up and deliveries from depots with lasso solutions. In *13th Nordic Logistics Research Network - NOFOMA 2001*, Reykjavik, Iceland, 2001.

[85] P. Hansen and N. Mladenovic̀. An introduction to variable neighborhood search. In S. Voss (Ed.), editor, *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Dordrecht, 1999.

[86] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

[87] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95) ; Vol. 1*, pages 607–615, Montréal, August 20-25 1995.

[88] A. Hoff and A. Løkketangen. Creating lasso-solutions for the traveling salesman problem with pickup and delivery by tabu search. *Central European Journal of Operations Research*, 14(2) :125–140, 2006.

[89] ILOG. Ilog solver reference manual, version 6.0.

[90] K. Inderfurth. Simple optimal replenishment and disposal policies for a product recovery system with leadtimes. *OR Spectrum*, 19 :111–122, 1997.

[91] P. Jaillet, J.F. Bard, L. Huang, and M. Dror. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science*, 32 :189–203, 1998.

[92] D.S. Johnson and C.H. Papadimitriou. Computational complexity. In A.H. G. Rinnooy Kan E. L Lawler, J. K. Lenstra and D. B. Shmoys (eds), editors, *The Traveling Salesman Problem*. John Wiley & Sons Chichester, 1985.

[93] B. Kalantari, A.V. Hill, and S.R. Arora. An algorithm for the traveling salesman problem with pick-up and delivery customers. *European Journal of Operational Research*, 22 :377 – 386, 1985.

[94] N. Karmarkar. A new ploynomial-time algorithm for linear programming. *combinatorica*, 4 :373 – 395, 1984.

[95] L. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20 :191 – 194, 1979.

[96] G.P. Kiesmüller. A new approach for controlling a hybrid stochastic manufacturing/remanufacturing system with inventories and different leadtimes. *European Journal of Operational Research*, 147 :62–71, 2003.

[97] G.P. Kiesmüller and C.W. Scherer. Computational issues in a stochastic finite horizon one product recovery inventory model. *European Journal of Operational Research*, 146 :553–579, 2003.

[98] J.U. Kim and Y.D. Kim. A decomposition approach to a multi-period vehicle scheduling problem. *International Journal of Management Science*, 27 :421–430, 2002.

[99] G. Kontoravdis and J.F. Bard. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7 :10–23, 1995.

[100] H.R. Krikke, A. Van Harten, and P.C. Schuur. On a medium term product recovery and disposal strategy for durable assembly products. *International Journal of Production Research*, 36(1) :111–139, 1998.

[101] H.R. Krikke, A. Van Harten, and P.C. Schuur. Business case Roteb : recovery strategies for monitors. *Computers & Industrial Engineering*, 36 :855–869, 1999.

[102] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, page 28, 1960.

[103] G. Laporte and F. Semet. Classical heuristics for the capacitated VRP. In *The Vehicle Routing Problem*, pages 109–128. SIAM, Philadelphia, PA, USA, 2001.

[104] H.C. Lau and Z. Liang. Pickup and delivery with time windows : Algorithms and test case generation. In *13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01)*, Dallas Texas, 2001.

[105] L.H. Lee, K.C. Tan, K. Ou, and Y.H. Chew. Vehicle capacity planning system : A case study on vehicle routing problem with time windows. *IEEE Transactions on Systems*, 33(2) :169–178, 2003.

[106] Z. Lu. *Planification hiérarchisée et optimisation des systèmes logistiques avec flux inverses*. PhD thesis, Université de Nantes, France, 2003.

[107] Z. Lu, N. Bostel, and P. Dejax. The simple plant location problem with reverse flows. In O.Zaikin A. Dolgui, J. Soldek, editor, *Supply Chain Optimization*. Kluwer Academic Publishers, 2004.

[108] O.B.G. Madsen. Lagrangean relaxation and vehicle routing. Technical Report IMSOR, The Technical University of Denmark, 1990.

[109] F. Malca and F. Semet. A tabu search heuristic for the pickup and delivery problem with time windows and a fixed size fleet. In *ORBELL 18*, Bruxelles, Belgique, 2004.

[110] V. Malépart, F. Boctor, J. Renaud, and S. Labillois. Nouvelles approches pour l'approvisionnement des stations d'essence. *Revue Française de Gestion Industrielle*, 22(2) :15–31, 2003.

[111] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulations and traveling salesman problems. *Journal of the Association for Computing Machinery*, 7 :326–329, 1960.

[112] S. Minner and R. Kleber. Optimal control of production and remanufacturing in a simple recovery model with linear cost functions. *OR Spektrum*, 23 :3–24, 2001.

[113] M. Minoux. *Résolution des problèmes de grandes dimensions : programmation linéaire généralisée et techniques de décomposition*. Programmation mathématiques, Théorie et Algorithmes, Tome 2. Dunod, 1983.

[114] S. Mitrovic-Minic. Pickup and delivery problem with time windows : A survey. Technical Report SFU CMPT TR 1998-12, Simon Fraiser University - Computing Science, 1998.

[115] G. Mosheiov. Vehicle routing with pick-up and delivery : tour-partitioning heuristics. *Comput. Ind. Eng.*, 34(3) :669–684, 1998.

[116] B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence 5*, 5 :188–224, 1989.

[117] G. Nagy and S. Salhi. Heuristic algorithm for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162 :126–141, 2005.

[118] I. Or. *Travelling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking*. PhD thesis, Department of Industrial Engineering and Management Sciences, 1976.

[119] G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *Operations Research Spectrum*, 27 :21 – 41, 2005.

[120] S.N. Parragh, K.F. Doerner, and R.F. Hartl. A survey on pickup and delivery problems. part I : Transportation between customers and depot. *Journal für Betriebswirtschaft*, to appear.

[121] S.N. Parragh, K.F. Doerner, and R.F. Hartl. A survey on pickup and delivery problems. part II : Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, to appear.

[122] K.D. Penev and A.J. De Ron. Determination of a disassembly strategy. *International Journal of Production Research*, 34(2) :495–506, 1996.

[123] L. Pitsoulis and M. Resende. Greedy randomized adaptive search procedures. In P.M.Pardalos and M.G.C.Resende, editors, *Handbook of Applied Optimization*, pages 168–181. 2001.

[124] J.Y. Potvin and J.M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66 :331–340, 1993.

[125] H.G.M Pullen and M.H.J Webb. A computer application to a transport scheduling problem. *Computer Journal*, 10 :10–13, 1967.

[126] M. Reiman, R. Rubio, and L.M. Wein. Heavy traffic analysis of the dynamic stochastic inventory routing problem. *Transportation Science*, 33(4) :361–372, 1999.

[127] J.-C. Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 362 – 367, 1994.

[128] C.C. Ribeiro. GRASP : Une métaheuristique gloutone et probabiliste. In J. Teghem and M. Pirlot, editors, *Optimisation approchée en recherche opérationnelle*, pages 153–176. Hermès, 2002.

[129] K. Ritcher. The extended eoq repair and waste disposal model. *International Journal of Production Economics*, 45(1–3) :443–448, 1996.

[130] D.S. Rogers and R.S. Tibben-Lembke. Going backwards : Reverse logistics trends and practices. *Reverse Logistics Executive Council*, 1999.

[131] L.M. Rousseau, M. Gendreau, and D. Feillet. Interior Point Stabilization for Column Generation. *Operations Research Letters*, 35(5) :660–668, 2007.

[132] L.M. Rousseau, M. Gendreau, and G. Pesant. Solving small VRPTWs with Constraint Programming Based Column Generation. In *CP-AI-OR'02*, mars 2002.

[133] L.M. Rousseau, M. Gendreau, G. Pesant, and F. Focacci. Solving VRPTWs with Constraint Programming Based Column Generation. *Annals of Operations Research*, 130 :199–216, 2004.

[134] N. Rudi, D. F. Pyke, and P. O. Sporsheim. Product recovery at the Norwegian National Insurance Administration. *Interfaces*, 30 :166–179, 2000.

[135] D. Sabin and E. Freuder. Contradiction conventional wisdom in constraint satisfaction. In editor Alan Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of Lecture Notes in Computer Science, PPCP'04 : Second International Workshop, Orcas Island, Seattle, USA, May 1994. Springer.

[136] M.W. Savelsbergh and M. Sol. The general pick-up and delivery problem. *Transportation Science*, 29(1) :17–29, 1995.

[137] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP '98)*, pages 417–431. Springer-Verlag, avril 1998.

[138] Y. Shen, J.Y Potvin, J.M Rousseau, and S. Roy. A computer assistant for vehicle dispatching with learning capabilities. *Annals of operations research,*, 61 :189–212, 1995.

[139] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2) :254–265, 1987.

[140] M. M. Solomon. Algorithms for the vehicle routing problem with time windows. *Transportation Science*, 29(2) :156–166, 1995.

[141] T. Spengler. Management of Material Flows in Closed-Loop Supply Chains : Decision Support System for Electronic Scrap Recycling Companies. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*, 2003.

[142] T. Spengler, H. Püchert, T. Penkuhn, and O. Rentz. Environmental integrated production and recycling management. *European Journal of Operational Research*, 97 :308–326, 1997.

[143] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31 :170–186, 1997.

[144] K. Taleb and S.M. Gupta. Disassembly of multiple product structures. *Computers and Industrial Engineering*, 32(4) :949–961, 1997.

[145] C. Tan and J. Beasley. A heuristic algorithm for the period routing problem. *Omega International Journal of Management Science*, 12(5) :497–504, 1984.

[146] D. Taqa Allah, J. Renaud, and F. Boctor. Le problème d'approvisionnement des stations d'essence. *Journal Européen des Systèmes Automatisés*, 34(1) :11–33, 2000.

[147] R. Teunter. Economic ordering quantities for recoverable item inventory systems. *Naval Research Logistics*, 48 :484–495, 2001.

[148] M.C. Thierry. *An analysis of the impact of product recovery management on manufacturing companies*. PhD thesis, Erasmus University, Rotterdam, 1997.

[149] M.C. Thierry, M. Salomon, J.A.E.E. Van Nunen, and L.N. Van Wassenhove. Strategic production and operations management issues in product recovery management. Technical Report Management Report Series No. 145, Erasmus University/Rotterdam school of Management, the Netherlands, 1993.

[150] M.C. Thierry, M. Salomon, J.A.E.E. Van Nunen, and L.N. Van Wassenhove. Stratégic issues in product recovery management. *California Management Review*, 37(2) :114–135, 1995.

[151] L.B. Toktay, L.M. Wein, and S.A. Zenios. Inventory management for remanufacturable products. *Management Science*, 46 :1412–1426, 2000.

[152] P. Toth and D. Vigo. Heuristics algorithms for the handicapped persons transportation problem. *Transportation Science*, 31(1), February 1997.

[153] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, USA, 2002.

[154] P. Trudeau and M. Dror. Stochastic inventory routing : Route design with stockouts and route failures. *Transportation Science*, 26 :17–184, 1992.

[155] A. Van Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer related, and time-related constraints*. PhD thesis, University of Antwerp, 1994.

[156] L.J.J. Van-Der-Bruggen, J.K. Lenstra, and P.C. Schuur. Variable-depth search for the single vehicle pickup and delivery problem with time windows. *Transportation Science*, 27(3) :298 – 311, 1993.

[157] E.A. Van Der Laan. *The effects of remanufacturing on inventory control*. PhD thesis, Erasmus University, Rotterdam, The Netherlands, 1997.

[158] E.A. Van Der Laan and M. Salomon. Production planning and inventory control with remanufacturing and disposal. *European Journal of Operational Research*, 102 :264–278, 1997.

[159] F. Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1) :111–128, 2000.

[160] N. Velasco, P. Dejax, , and C. Guéret. Une approche par génération de colonnes pour un problème de tournées d´hélicoptères. In *Congrès ROADeF'05*, février 2005.

[161] D. Vlachos and R. Dekker. Return handling options and order quantities for single period products. *European Journal of Operational Research*, 151 :38–52, 2003.

[162] C. Voudouris. *Guided Local Search for Combinatorial Problems*. PhD thesis, University of Essex, 1997.

[163] C. Voudouris and E. Tsang. Guided local search. Technical Report Technical Report CSM-247, Department of Computer Science, University of Essex, 1995.

[164] M. Witurcki, P. Dejax, and M. Haouari. Un modèle et un algorithme de résolution exacte pour le problème de tournées de véhicules multi périodiques - une application à la distibution des gaz industriels. In *Congrès Franco - Québécois de Génie Industriel*, 1997.